Development of a web application for weekly meal planning

Kier Davis

28th March 2014

Contents

List of Tables

List of Figures

List of Code Listings

1. Analysis of Problem

1.1 Background and identification of problem

Every week our family plans the meals we will be cooking and eating during the week. The task is organised by my mother, Steph. This process takes longer than she would like because she often gets stuck for ideas and everyone in the family has different preferences.

The primary problem with this process is that she must rely on her memory and her family members' memories in order to find meals to cook. No organised list of all meals currently exists to draw ideas from, meaning that often only a small number of meals are consistently remembered while others remain forgotten. The creation of a system with a formalised database of meals and a way to automatically make suggestions for meals would greatly help to rectify this.

1.2 Description of the current system

The current system is entirely a manual one. Steph first asks the members of her family to provide ideas of meals to serve for the week, although they are often found to be uncooperative. She then uses these suggestions to come up with a meal plan, listing which meal will be served on each day.

Certain criteria restrict which meals can be served on each day; these include stopping similar meals from being allocated to nearby days, and the same meal from being allocated too often, in order to increase variety of meals. Other factors must also be considered in the decision, such as the time taken to cook each meal and whether the meal is liked or disliked by the family as a whole. The meal plan is then used to formulate a shopping list; once the ingredients have been purchased the meals are finally cooked and served.

1.3 Identification of the prospective user

The principal user of this system will be Steph, because she leads the current process and is responsible for the majority of the shopping and cooking. The system could also be used by anyone who has this requirement to plan meals.

1.4 Identification of user needs and acceptable limitations

An interview was conducted to determine what the requirements and acceptable limitations of the project are; the transcript of this is given in Appendix A (p. 59).

Primarily, my user feels that the main problems with the current system are

- the lack of a list of meals that could be served, leading to some meals being forgotten if they are not served in a while.
- the fact that she finds it difficult to come up with ideas for meals to serve.

My user requires me to create a system that will allow her to:

- add, edit and remove meals within a database.
- store a hyperlink to a recipe for each meal.
- categorise or tag meals according to their primary ingredients.
- search for meals by name and/or category.
- rate meals as generally liked or disliked.
- view an automatically generated list of suggested meals, based on variety and how much the meal is enjoyed by the family.
- manually construct a meal plan from this list of suggestions.
- view an automatically generated meal plan.
- print either of the above meal plans.
- use the application on her mobile phone.

The system is not required to:

- allow family members to specify individual food preferences.
- automatically generate shopping lists.
- be multi-user.

1.5 Description of the proposed system

The proposed system must be able to store a list of meals in a database, which can be added to, edited and removed from by the user. Similarly, a list of meal plans will also be stored, which can be added to, edited and removed from. A meal plan will be defined by the start and end dates for the range of consecutive days it covers. A serving of a meal will be represented as a link between the meal plan it is in, the meal being served and the date on which it is served.

1.6 Data sources and destinations

The data sources and destinations for the proposed system are listed in Table 1.1 (p. 13).

1.7 Data volumes

The database will store around 50 meals (but this number may be added to over time). Each meal will likely have no more than 10 tags: a total of approximately 500 tags. However, many tags will be shared between meals and it would be rare for meals to have more than 5 tags, making this number a large overestimate.

The database will have capacity for 100 meal plans, which can be increased for example by changing an application setting. Each meal plan will be for on average 7 days, meaning that 700 meal serving records will need to be stored.

A summary of these statistics is listed in Table 1.2 (p. 13).

1.8 Data dictionary

The data dictionary for the proposed system is presented in Table 1.3 (p. 14).

1.9 Data flow diagrams

Data flow diagrams for the current system are shown in Figure 1.1 and Figure 1.2 (p. 13).

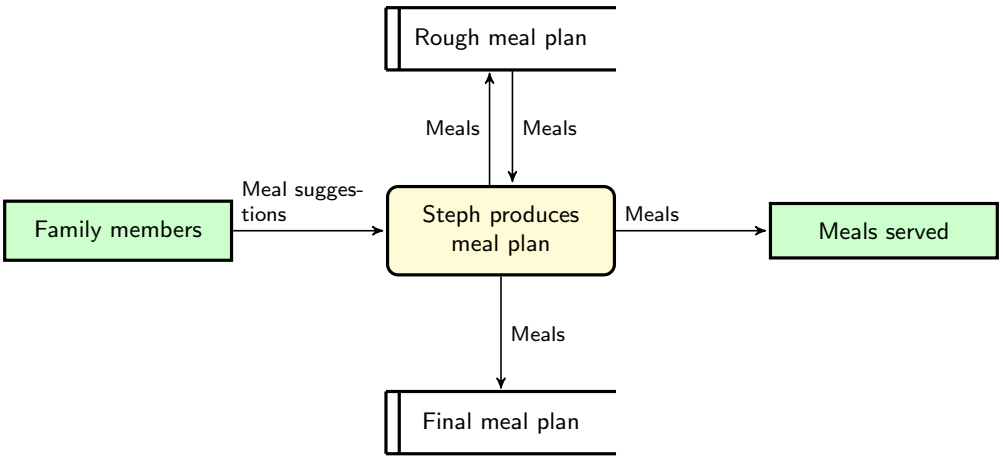

Figure 1.1 – Level 0 (contextual) data flow diagram for current system

Data flow diagrams for the proposed system are shown in Figure 1.3 (p. 14) and Figure 1.4 (p. 15).

Data	Source	Destination
Meals	Steph's memory, recipe books	Database of meals
Meals	Database of meals	Meal plan editor
Meal suggestions	Recommendation algorithm	Meal plan editor
Meal plans	Meal plan editor	Printer, meal plan database

Table 1.1 – Data sources and destinations in proposed system

Entity	Approx. number of records
Meal	50
Tag	500
Meal plan	100
Meal serving	700

Table 1.2 – Summary of data volumes

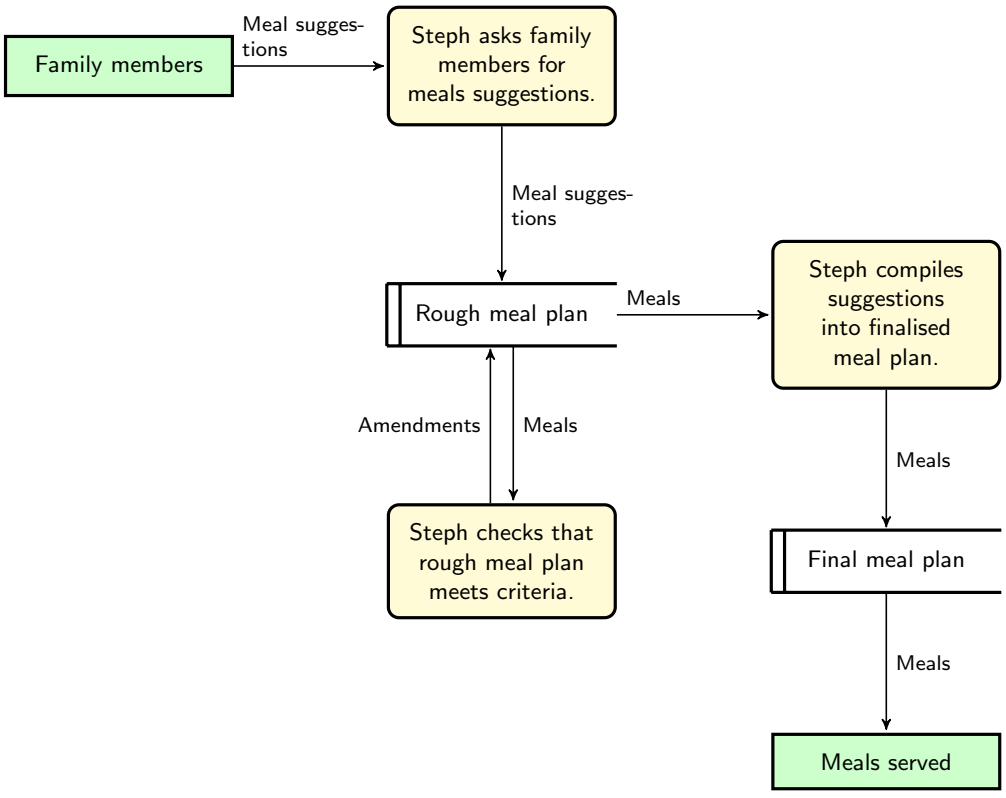

Figure 1.2 – Level 1 data flow diagram for current system

Entity	Field name	Description	Type	Example data
Meal	Meal name	The name/title of a meal.	Text	Chilli con carne
	Recipe URL	The URL of a recipe or other information associated with a meal.	Text	http://...
	Tag(s)	A keyword or category used to classify meals into groups.	Text	pasta
	Favourite	A mark of whether the meal is a favourite (and so appears higher on the suggestion list).	Boolean (yes/no)	yes
	Score	A numeric indication of how much a meal is recommended at a particular time.	Number	5.27
Meal plan	Notes	Notes that the user may want to associate with the meal plan.	Text	Eating out w/ friends on Tue
	Start date	The first day in the series of consecutive days associated with this meal plan.	Date	2013-10-14
	End date	The last day in the series of consecutive days associated with this meal plan.	Date	2013-10-18
Meal serving	Date	The date of the day that this serving of a meal is on.	Date	2013-10-15

Table 1.3 – Dictionary of data to be stored in the proposed system

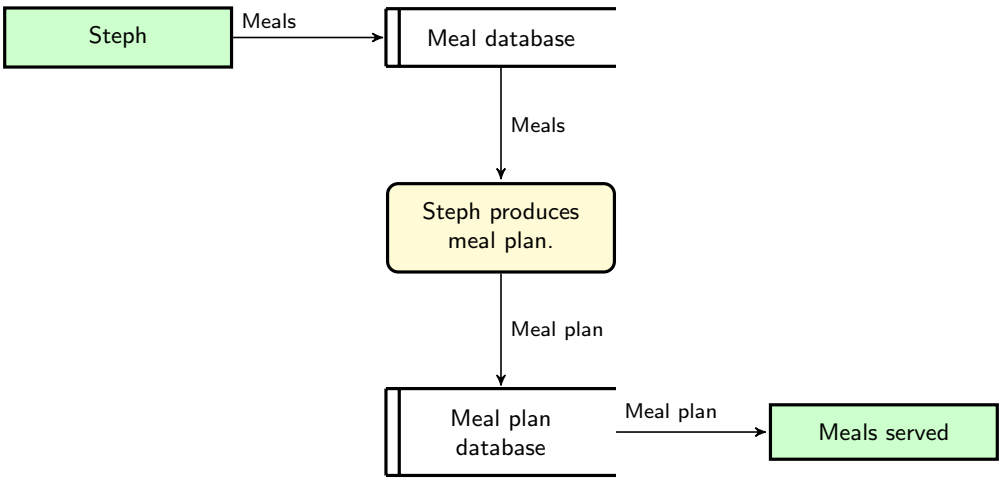

Figure 1.3 – Level 0 (contextual) data flow diagram for proposed system

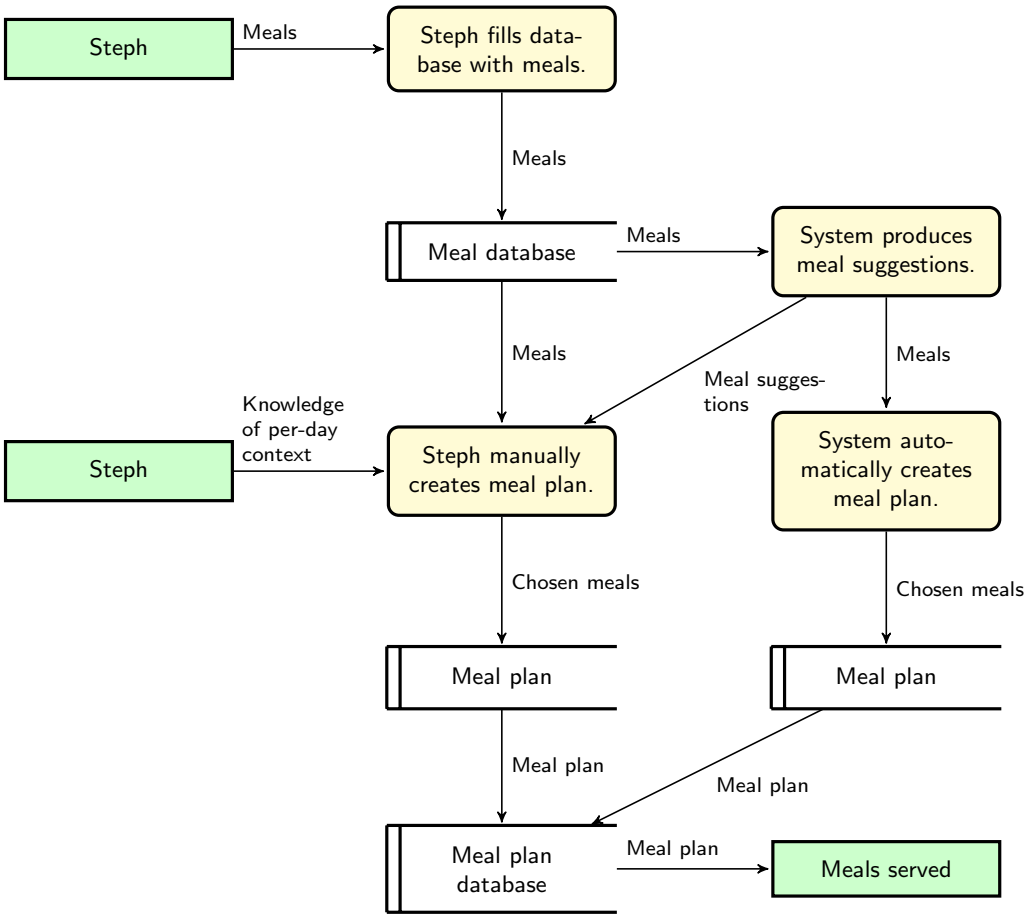

Figure 1.4 – Level 1 data flow diagram for proposed system

1.10 Entity relationship model

The entity relationship model of the proposed system is presented in Figure 1.5.

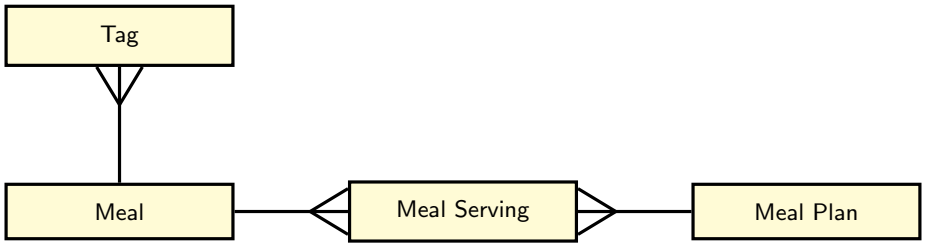

Figure 1.5 – Entity relationship model of the proposed system

1.11 Project objectives

The system must:

1. allow the user to fully manage a collection of meals, and generate suggestions based on other servings of meals.
2. allow the user to add a meal to the database, supplying a name, hyperlink to a recipe and one or more tags.
3. allow the user to update the same three pieces of information as in objective 2 stored about a meal.
4. allow up to 100 meals to be stored in the database, with room for expansion.
5. allow a meal to be marked as favourited or not.
6. allow the user to remove meals from the database.
7. allow the user to search for meals by name or by tag. The search operation must complete in under 0.1 seconds for a database of 50 meals, excluding network transmission delays. The view should be paginated.
8. allow the user to browse all meals in the database. The view should be paginated, and the user must be able to sort the results by the default order (order of primary key), by alphabetical order of name or by "score" (a numerical quantity calculated based on the suggestion criteria).
9. allow the user to browse all meals in the database that have a specific tag. The two additional criteria in objective 8 also apply here.
10. be able to automatically create a meal plan, which is stored in the database. The meal plan generation should take no more than 0.5 seconds.
11. allow the user to manually create a meal plan, assisted by the database views described in objectives 7, 8 and 9. This meal plan must be able to be saved in the database.
12. allow up to 100 meal plans to be stored in the database, with room for expansion.

13. allow the user to browse the past meal plans stored in the database. The view should be paginated.
14. allow the user to update the meal plans stored in the database. The updatable information should include the meal servings associated with the meal plan as well as the start and end dates.
15. ask the user to confirm when changing the start or end date of a meal plan would leave meal servings outside of the range. If the user does confirm, these invalidated meal serving records must be deleted.
16. allow the user to remove meal plans from the database.
17. allow the user to print meal plans stored in the database.
18. provide web page views suitable for mobile devices as well as those suitable for desktop computers.

1.11.1 Authentication

These objectives accurately outline the goals of the project and represent the views and opinions of the client.

Signature of client: .....

1.12 Comparison of potential solutions

A comparison of four potential solutions is given in Table 1.4 (p. 18).

To choose a solution from these, the user's requirements must be foremost. It is important that the chosen solution can automatically produce meal suggestions. Solution one can hold a database of meals, but cannot perform complex operations on this data beyond simple searching and so is unable to produce suggestions. Solution two offers more than solution one in terms of filtering and sorting the data, and could produce suggestions by sorting on a "score" column in the spreadsheet. However, solution two cannot store multiple tags for one meal, or multiple meals for one meal plan without losing the ability to efficiently sort and search on this data; this is caused by spreadsheets being non-relational databases. For these reasons, solutions one and two are ruled out.

This leaves solutions three and four as possible candidates. They are identical in all aspects except for the medium in which they are delivered to the user (as a desktop application or as a web application). Because it can be accessed from multiple devices (one of the features the user has requested), and because I am more familiar with the technology that would be used to implement it, I have chosen solution four over solution three.

A major benefit of choosing solution four is that is can be built directly to the user's needs, ensuring that the application will behave exactly as the user intends and removing the obstacle

Suggested solution	Advantages	Disadvantages
A paper-based diary for recording which meals are served.	• Does not require a computer to be used. • Simple and intuitive to use. • Can be flicked through to find ideas for meals.	• Difficult to filter and sort. • Cannot automatically produce suggestions. • Cannot easily store a link to a recipe.
A Microsoft Excel spreadsheet-based database.	• The user is familiar with the Excel software. • Excel provides a variety of ways to filter and sort data. • Can make meal suggestions.	• Tied to a single device. • Not a relational database - cannot efficiently store a complex data model.
A bespoke Windows desktop application.	• Tailored to user's exact needs. • Easy to filter and sort data. • Can make meal suggestions automatically. • The user is familiar with Windows desktop-style applications.	• Requires time to be developed. • Tied to a single device. • The programmer is not familiar with developing Windows desktop applications.
A bespoke web application.	• Tailored to user's exact needs. • Easy to filter and sort data. • Can make meal suggestions automatically. • Can be accessed from many devices. • The programmer is relatively familiar with developing web applications.	• Requires time to be developed.

Table 1.4 – Realistic comparison of potential solutions

of the user not being familiar with this style of application. It will also support relational storage of data whilst providing powerful searching, sorting and filtering facilities over it. The user will be able to access it from her phone as well as her computer, which is one of the requirements of the project. Finally, I have experience developing web applications, which means that less time will be required to learn the tools that will be used to build it.

2. Design of Solution

2.1 Overall system design

The system will be implemented as a web application consisting of a number of dynamically generated web pages, or "views". A homepage will allow navigation to other areas of the application. The system will store in a database a list of meals and a list of meal plans, both of which can be browsed, added to, edited and removed from.

A module-based structure will be used to organise the system, seperating data processing algorithms from database manipulation routines and web page generation code. This will help to improve the clarity of the codebase. Additionally, only accessing the database through set, clearly defined procedures reduces the risk that the database may be manipulated inconsistently and aids in maintaining referential integrity.

A system flow chart for the system's functionality is given in Figure 2.1 (p. 21).

2.2 Description of technologies used

The system will be implemented as a web application written in Go[1] for server-side code and JavaScript for client-side code. The web pages are to be presented using HTML and CSS, which are the standard web technologies for this task.

Unlike static websites, this application will implement a web server program rather than writing scripts for an existing web server. Go's built-in `net/http` package will be used as a framework to handle the HTTP protocol details, meaning that only the individual page handlers need to be written; similarly, the `html/template` package provides a simple templating system to allow HTML pages to be dynamically generated with minimal work by the programmer.

The Go programming language was chosen as the server-side language because:

- it is aimed towards web application development.
- it compiles directly to machine code, for efficient execution.
- it has a rich standard library, ranging from servers and clients for many network protocols to cryptographic and image-processing functions.
- it has language support for concurrent programming, making parallel programming easier as well as improving code structure (by separating code performing different tasks out into completely different threads of control).
- its compiler encourages well-written code.
- I am familiar with the language's syntax and standard libraries.

[1] See `http://golang.org/`

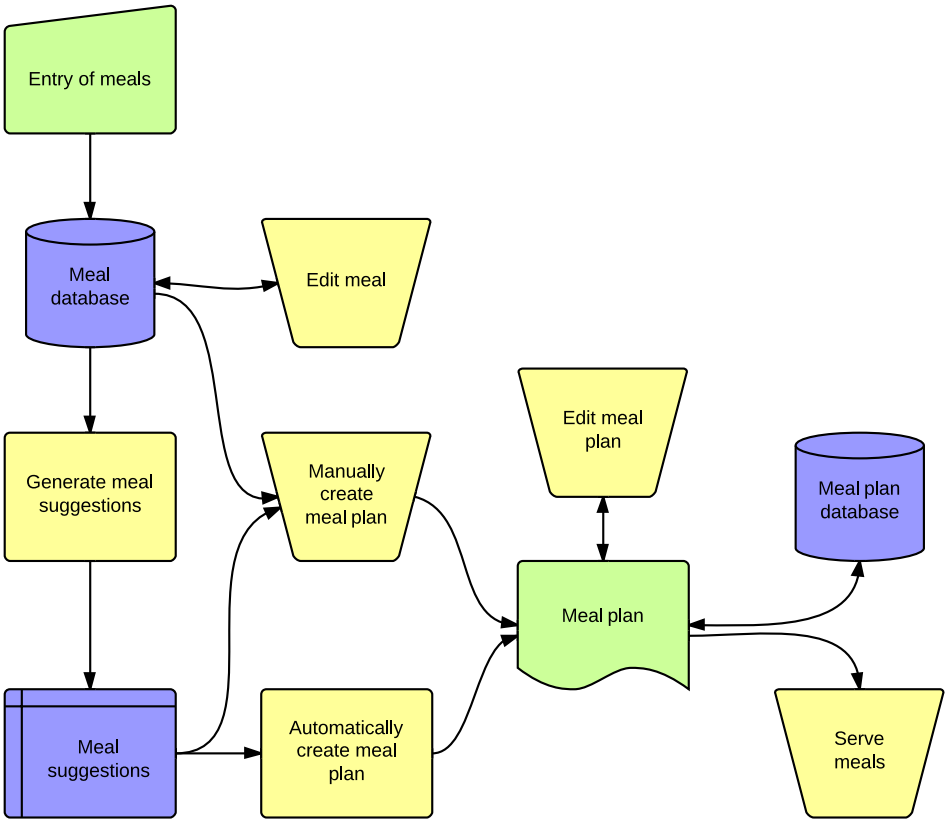

Figure 2.1 – System flow chart for the system

Most of the functionality of the web application will be performed by HTTP `POST` requests generated by submission of forms; however, operations that need not direct the user to a different page are implemented using asynchronous API calls made by client-side code - a practice known as AJAX[2].

The database engine that will be used for the system will be MySQL[3], a popular SQL-based database management system; however, the code will be written portably such that the database engine could be replaced with another (for example SQLite) with minimal changes to the code. A relational database was chosen over a flat file because the system requires one-many relationships, which are hard to implement in a flat file. To connect to the database from the application, the third-party `Go-MySQL-Driver`[4] package will be used.

2.3 Modular system structure

The modular structure of the system is given in Figure 2.2 (p. 23) and Figure 2.3 (p. 23).

[2]Asynchronous JavaScript and XML; see `http://en.wikipedia.org/wiki/Ajax_(programming)`
[3]See `http://www.mysql.com/`
[4]See `https://github.com/go-sql-driver/mysql`

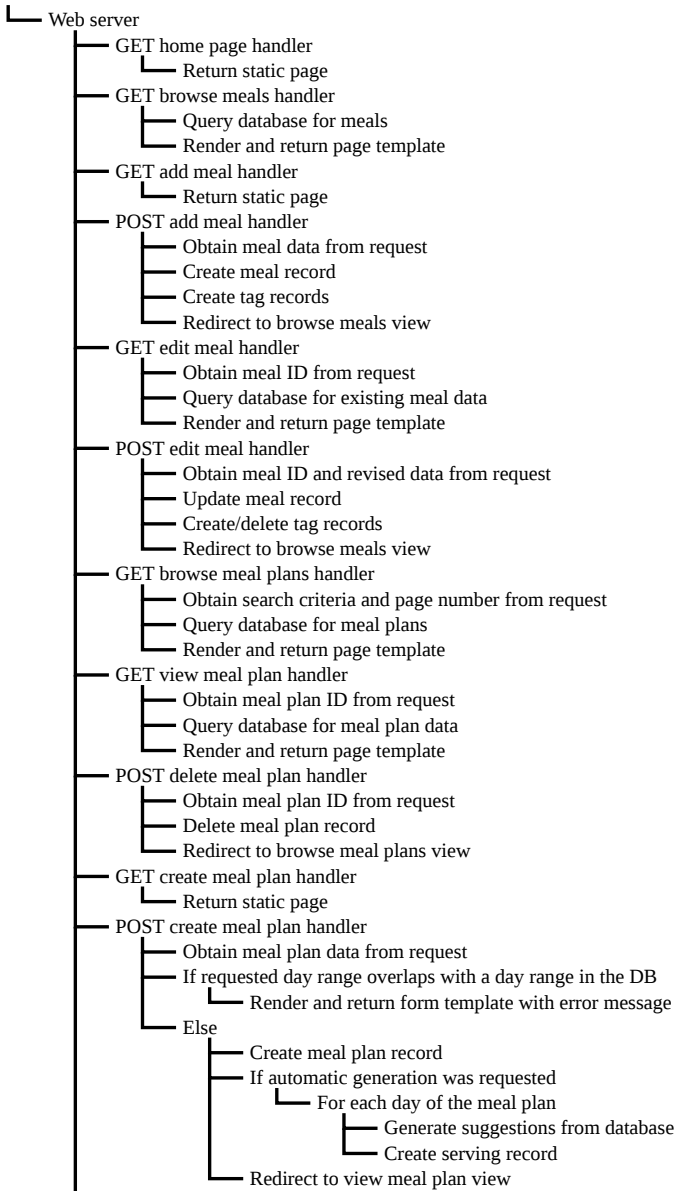

Figure 2.2 – Modular structure for the system (part 1)

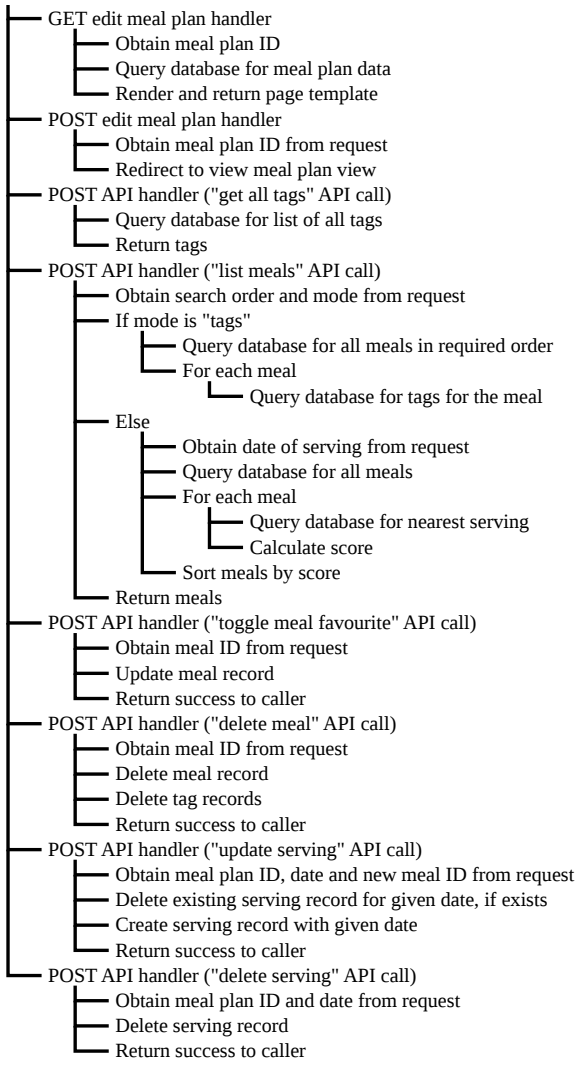

Figure 2.3 – Modular structure for the system (part 2)

2.4 Database tables

The database tables that will be used in the system are outlined in Table 2.1, and the validation for these fields is described in Table 2.2 (p. 25). An entity relationship diagram for these relations was given earlier in Figure 1.5 (p. 16).

Table	Field	Description	Data type
`meal`	`id`	The unique identifier for the meal.	64 bit unsigned integer
	`name`	The name/title of the meal.	String (up to 255 chars)
	`recipe`	The URL of a recipe or other information associated with the meal.	String (unbounded)
	`favourite`	Whether or not the user has marked this as a favourite meal.	Boolean
`tag`	`mealid`	The identifier of the meal being tagged.	64 bit unsigned integer
	`tag`	The tag.	String (up to 64 chars)
`mealplan`	`id`	The unique identifier for the meal plan.	64 bit unsigned integer
	`notes`	Notes associated with the meal plan.	String (unbounded)
	`startdate`	The first day in the series of consecutive days associated with the meal plan.	Date
	`enddate`	The last day in the series of consecutive days associated with the meal plan.	Date
`serving`	`mealplanid`	The identifier of the meal plan the meal is served on.	64 bit unsigned integer
	`dateserved`	The date that this serving of a meal is on.	Date
	`mealid`	The identifier of the meal being served.	64 bit unsigned integer

Table 2.1 – Database tables to be used in the system

Table	Field	Validation
`meal`	`id`	Must be unique.
	`name`	Must be 255 characters or fewer.
	`recipe`	If not null, must be a valid URL.
	`favourite`	No validation required.
`tag`	`mealid`	Must point to a valid record in the `meal` table.
	`tag`	Must be 64 characters or fewer. The combination of this field and the `mealid` field must be unique.
`mealplan`	`id`	Must be unique.
	`notes`	No validation required.
	`startdate`	Must be a valid date.
	`enddate`	Must be a valid date. Must be a date occuring on or after `startdate`.
`serving`	`mealplanid`	Must point to a valid record in the `mealplan` table.
	`dateserved`	Must be a valid date. Must be a date occuring between the `startdate` and `enddate` of the meal plan pointed to the `mealplanid` field, inclusive. The combination of this field and the `mealplanid` field must be unique.
	`mealid`	Must point to a valid record in the `meal` table.

Table 2.2 – Database field validation to be applied to the system

2.5 Algorithms

2.5.1 Meal scoring algorithm

An algorithm to produce the score for a meal is given in Listing 2.1.

The calculation used to produce the initial value for *score* is $1.35 - \frac{2.8}{distance+1}$. This particular mathematical function was chosen since it provides an ideal graph shape—the further the serving is from the day being edited, the smaller the effect changing this distance would have on the score. A graph for this function is given in Figure 2.4 (p. 27).[5]

The algorithm makes use of the following functions:

DateDiff Takes two dates and returns the number of days between them, where identical dates are considered to have 0 days between them, consecutive dates to have 1 and so on.

FindClosestServing A database operation implemented by the SQL code in Listing 2.8 (p. 30). It takes a meal identifier and a date and returns the date of the closest serving of that meal to that date.

```
function CalcScore(mealID, isFavourite, dateBeingEdited)
    closestServingDate ← FindClosestServing(mealID, dateBeingEdited)
    distance ← DateDiff(dateBeingEdited, closestServingDate)
    score ← 1.35 − (2.8 ÷ (distance + 1))
    if isFavourite then
        score ← score × 2
    end if
    return score
end function
```

Listing 2.1 – Pseudocode algorithm to produce a score for a meal

2.5.2 Meal list pagination algorithm

An algorithm to select the correct range of meals to show on a particular page of results is given in Listing 2.2 (p. 27).

The algorithm makes use of the following functions:

Error Causes an exception to occur with the given message.

Length Takes a list and returns the length of the list.

SubRange Takes a list and two indices within the list, and returns the subrange of that list beginning at the first index and ending just before the second index.

[5] Graph generated by FooPlot; `http://fooplot.com/`

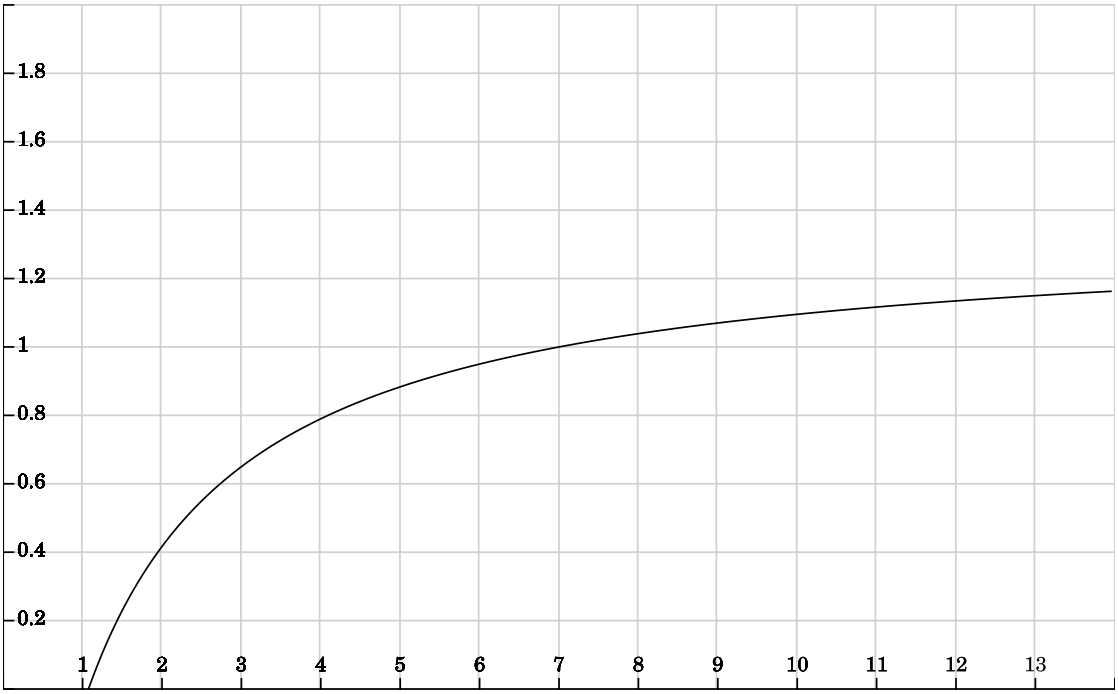

Figure 2.4 – Graph of $y = 1.35 - \frac{2.8}{x+1}$

```
function GetMealPage(meals, pageNumber)
    numMeals ← Length(meals)
    startIndex ← (pageNumber − 1) × 10
    if startIndex >= numMeals then
        Error("Pagenumbertoohigh")
    end if
    stopIndex ← startIndex + 10
    if stopIndex >= numMeals then
        stopIndex ← numMeals − 1
    end if
    return SubRange(meals, startIndex, stopIndex)
end function
```

Listing 2.2 – Pseudocode algorithm to select a page of meals from the entire list

2.6 Database queries and manipulations

2.6.1 Database creation

The DDL statements required to create the database and its tables are given in Listing 2.3.

```
CREATE DATABASE mealplanner;

CREATE TABLE meal (
    id          BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name        VARCHAR(255) NOT NULL,
    recipe      TEXT,
    favourite BOOLEAN NOT NULL,
   PRIMARY KEY (id)
);

CREATE TABLE tag (
    mealid BIGINT UNSIGNED NOT NULL,
    tag     VARCHAR(64) NOT NULL,
   PRIMARY KEY (mealid, tag)
);

CREATE TABLE mealplan (
    id          BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    notes       TEXT,
    startdate DATE NOT NULL,
    enddate     DATE NOT NULL,
   PRIMARY KEY (id)
);

CREATE TABLE serving (
    mealplanid BIGINT UNSIGNED NOT NULL,
    dateserved DATE NOT NULL,
    mealid      BIGINT UNSIGNED NOT NULL,
   PRIMARY KEY (mealplanid, dateserved)
);
```

Listing 2.3 – SQL statements to create the database and tables

2.6.2 Meal browsing

The meal browser view and the meal plan editor view require a list of meals to be obtained via an API call. The sort order be changed by the user, so it is passed as a parameter. Another parameter passed to the API call is the mode; this is either “tags” to look up the tags for each meal as well (used in the meal browser view) or “suggestions” to generate meal suggestions and produce a rating for each meal (used in the meal plan editor view).

A DML statement to fetch the search results:

- in unsorted order (the order the records were added) is given in Listing 2.4.
- in alphabetical order of name is given in Listing 2.5.

```
SELECT meal.id, meal.name, meal.recipe, meal.favourite
FROM meal;
```

Listing 2.4 – SQL statement to list all meals in the database without sorting

```
SELECT meal.id, meal.name, meal.recipe, meal.favourite
FROM meal
ORDER BY meal.name ASC;
```

Listing 2.5 – SQL statement to list all meals in the database in alphabetical order

To produce meal suggestions (given the date to generate a serving for), the algorithm given earlier in Section 2.5.1 (p. 26) is used to calculate a score for each meal. The scores are written back into the database in a temporary table, and then the list of meals is sorted by cross-referencing the meals with the scores. The process to do this is given by the following steps:

1. A temporary table is created. A DDL statement for this is given in Listing 2.6 (p. 30).
2. The list of meal IDs and "favourite" statuses (the information needed to calculate scores) is fetched. A DML statement for this is given in Listing 2.7 (p. 30).
3. For each meal, the score is calculated. This involves finding the date of the serving of a meal with the lowest proximity to a particular date. If the meal is served on the given date then this serving is ignored (since it would be overwritten when the serving is updated). A DML statement template for this is given in Listing 2.8 (p. 30).
4. Calculating the score also involves finding all tags associated with meals near the date of the day being edited, and the proximity to them. A DML statement template to perofrm this is given in Listing 2.9 (p. 30).
5. Each score is inserted into the temporary table. This can be done in batches with the score generation (e.g. generate 20 scores, then insert 20 scores with one SQL command) in order to reduce overhead. A DML statement template for this is given in Listing 2.10 (p. 30).
6. The list of meals sorted by score is retrieved. A DML statement for this is given in Listing 2.11 (p. 30).
7. The temporary table is deleted. A DDL statement for this is given in Listing 2.12 (p. 32).

2.6.3 Getting info about a meal

Some views require the information about a specific meal to be obtained. A DML statement template for this is given in Listing 2.13 (p. 32).

```
CREATE TEMPORARY TABLE score (
  mealid BIGINT UNSIGNED NOT NULL,
  score  FLOAT NOT NULL
);
```

Listing 2.6 – SQL statement to create a temporary table to hold scores in

```
SELECT meal.id, meal.favourite
FROM meal;
```

Listing 2.7 – SQL statement to list all meal IDs and favourite statuses

```
SELECT serving.dateserved
FROM serving
WHERE serving.mealid = ≪Meal identifier≫
AND serving.dateserved != ≪Date to generate suggestions for≫
ORDER BY ABS(DATEDIFF(serving.dateserved,
  ≪Date to generate suggestions for≫)) ASC
LIMIT 1;
```

Listing 2.8 – SQL statement template to find the closest serving of a meal to a date

```
SELECT tag.tag, ABS(DATEDIFF(serving.dateserved,
  ≪Date being edited≫))
FROM serving
INNER JOIN tag ON tag.mealid = serving.mealid
WHERE ABS(DATEDIFF(serving.dateserved,
  ≪Date being edited≫)) <= 7
```

Listing 2.9 – SQL statement template to find all tags associated with meals served within 7 days of a given date

```
INSERT INTO score
VALUES (≪Meal identifier 1≫, ≪Score 1≫),
       (≪Meal identifier 2≫, ≪Score 2≫),
       /* ... */
       (≪Meal identifier n≫, ≪Score n≫);
```

Listing 2.10 – SQL statement template to insert a batch of meal scores into the temporary table

```
SELECT meal.id, meal.name, meal.favourite, score.score
FROM meal
INNER JOIN score ON score.mealid = meal.id
ORDER BY score.score DESC;
```

Listing 2.11 – SQL statement to list all meals sorted by score in descending order

The list of tags associated with a meal can also be found using the DML statement template given in Listing 2.14 (p. 32).

2.6.4 Meal addition

The addition of a meal to the database involves two steps. The first is the addition of a record to the `meal` table; a DML statement template for this is given in Listing 2.15 (p. 32). The `id` field is given the value `NULL`, since this value is ignored by the database engine and an automatically generated unique value is used instead. The `favourite` is given the default value 0 (for false).

The second step is the addition of a number of records to the `tag` table that link to the record created in the first step; a DML statement template for this is given in Listing 2.16 (p. 32).

2.6.5 Meal editing

The editing of a meal is divided into the following steps:

1. The meal record is updated. A DML statement template for this is given in Listing 2.17 (p. 32).
2. All tags previously associated with the meal are removed. A DML statement template for this is given in Listing 2.18 (p. 32).
3. The new set of tags is added to the meal. A DML statement template for this was given in Listing 2.16 (p. 32).

2.6.6 Meal plan creation

The creation of a meal plan consists of two parts overall; however the second part can be considered the same as editing a meal and is implemented this way in the user interface. The first part of meal plan creation involves the addition of a new record to the `mealplan` table. A DML statement template for this is given in Listing 2.19 (p. 33). The `id` field is again given the value `NULL` since this value is ignored and an automatically generated unique value is used instead.

2.6.7 Meal plan editing

The editing of a meal plan involves creating, updating and removing meal serving records. These tasks are performed via two API calls, “update serving” and “delete serving”. Updating a serving deletes the existing serving record if it already exists, and creates a new one. DML statement templates for these are given in Listing 2.20 (p. 33) and Listing 2.21 (p. 33).

```
DROP TABLE score;
```

Listing 2.12 – SQL statement to delete the temporary score table

```
SELECT meal.name, meal.recipe
FROM meal
WHERE meal.id = ≪Meal identifier≫;
```

Listing 2.13 – SQL statement template to fetch information about a meal

```
SELECT tag.tag
FROM tag
WHERE tag.mealid = ≪Meal identifier≫;
```

Listing 2.14 – SQL statement template to fetch tags associated with a meal

```
INSERT INTO meal
VALUES (
  /* id */          NULL,
  /* name */        ≪Title≫,
  /* recipe */      ≪Recipe URL≫,
  /* favourite */ 0
);
```

Listing 2.15 – SQL statement template to add a meal to the database

```
INSERT INTO tag
VALUES (≪Meal identifier≫, ≪Tag 1≫),
       (≪Meal identifier≫, ≪Tag 2≫),
       /* ... */
       (≪Meal identifier≫, ≪Tag n≫);
```

Listing 2.16 – SQL statement template to add tags to a meal in the database

```
UPDATE meal
SET meal.name = ≪Title≫,
    meal.recipe = ≪Recipe URL≫
WHERE meal.id = ≪Meal identifier≫;
```

Listing 2.17 – SQL statement template to update a meal in the database

```
DELETE FROM tag
WHERE tag.mealid = ≪Meal identifier≫;
```

Listing 2.18 – SQL statement template to remove all tags from a meal in the database

```
INSERT INTO mealplan
VALUES (
  /* id */          NULL,
  /* notes */       ≪Notes≫,
  /* startdate */ ≪Start date≫,
  /* enddate */     ≪End date≫
);
```

Listing 2.19 – SQL statement template to add a new meal plan to the database

```
DELETE FROM serving
WHERE serving.mealplanid = ≪Meal plan ID≫
AND serving.dateserved = ≪Date of meal plan slot≫;
```

Listing 2.20 – SQL statement template to delete a serving record from the database

```
INSERT INTO serving
VALUES (
  /* mealplanid */ ≪Meal plan ID≫,
  /* dateserved */ ≪Date of meal plan slot≫,
  /* mealid */       ≪Meal ID≫
);
```

Listing 2.21 – SQL statement template to insert a serving record into the database

2.7 Human-computer interface rationale

The pages for this web application will be designed to look professional, consistent and easy to understand. The Blueprint CSS toolkit [6] will be used as a base style that provides a clean look that displays the same on most major browsers.

Each page will have clearly presented hyperlinks to other pages in the application; the majority of these are in the form of an ancestor hierarchy or "breadcrumb trail" widget, which consists of a string of short page identifiers separated by arrows indicating the ancestors in a hierarchy of pages. The widget will be placed in the same location on all pages, so that the user can quickly find the widget to enable quick navigation. The hierarchy of pages is illustrated in Figure 2.5.

The add meal and edit meal views are two separate views; however they are grouped into one in this section since the page layout is identical, except for the title.

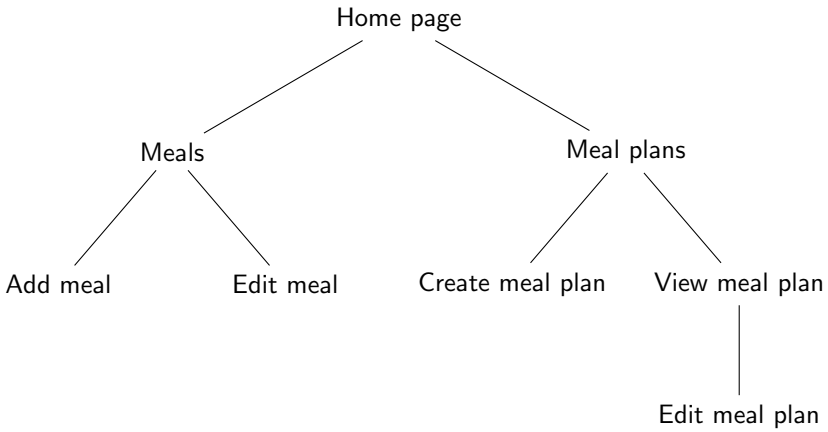

Figure 2.5 – Hierarchy of web page views in the application

2.7.1 Features common to all pages

- All pages have a title in large font in the top left corner, making it clear to the user which page they are viewing.
- All pages have an ancestor hierachy visible above the title of the page. By placing it in a consistent location on every page, the user will quickly learn where it is, allowing for efficient navigation of the site.
- The primary navigation link (e.g. a link to the add meal view on the browse meals view) will be placed in the top right corner of the page. The same effect applies as the above point.
- All headings will use the same serif font and all body text will use the same sans serif font; using too many different fonts can result in pages that look cluttered and confusing.

[6] See `http://www.blueprintcss.org/`

- Buttons will have icons where possible, to speed up the process of identification of the purpose of the button

2.7.2 Home page

Description This view will act as a "dashboard" for the user, with hyperlinks to the other views in the application.

Features The home page simply contains buttons that will direct the user to other parts of the application. The buttons and icons are large to make better use of the space and to draw attention to the four ways the user can begin using the application.

Incoming navigation The home page can be accessed quite simply, as it is the landing page of the website. Every other view in the application should contain a hyperlink to the home page; this is provided by the page hierarchy widget.

Outgoing navigation The home page contains links to the meal browser, meal plan browser, add meal and create meal plan views.

Mockup A mockup of this view is shown in Figure 2.6 (p. 39).

2.7.3 Browse meals

Description This view will allow the user to browse the list of meals stored in the database, including searching, sorting and deleting facilities. It will also have a hyperlink to the "add meal" view, as well as "edit meal" views for each meal.

Features The centrepoint of the meal browser view is the list of meals, shown in a table. Each row shows the name and tags of the meal, a link to the recipe associated with the meal and a number of "action buttons".

The first action button has the image of a heart and is a toggle button (where clicking toggles the state from off to on and back, like a check box) representing whether the meal is favourited. Clicking it performs a background API call to update the database without leaving the page.

The second action button has the image of a pencil and directs the user to an editing view for the meal.

The third action button has the image of a trash can and presents a confirmation dialogue box asking the user if they are sure they want to continue with deleting the meal. Selecting "yes" in this dialog deletes the meal from the database and presents a "meal deleted" dialogue box. This is performed by way of a background API call, removing the record from the database without leaving the page.

The table is presented with a pagination system, in order not to overload the user with information. A label centered below the table informs the user of the current page number and the total number of pages, while "first", "previous", "next" and "last" buttons are placed at the edges. The entire dataset is downloaded via an API call once the webpage is ready.

A search facility is also provided above the table; the search operation will commence when the user stop typing, eliminating the need for a search button and increasing response time

at the expense of having to perform redundant searches if the user pauses while typing their query. The searching of the dataset will be performed clientside.

Incoming navigation The meal browser view can be accessed from the home page. The add/edit meal view will also have links back to this page.

Outgoing navigation The meal browser view links back to the homepage and forward to the add meal view. It also contains a link to an edit meal view for each meal listed.

Mockup A mockup of this view is shown in Figure 2.7 (p. 40).

2.7.4 Add / edit meal

Description This view will have a form allowing the user to enter the details of a meal. Submitting this form adds these details to the database, either as a new meal record or amending an existing one.

Features At the top of the main body of this view are two text fields with associated labels. The fields are for the name and the recipe URL parts of the meal record. In the add meal view these will be initially empty, while in the edit meal view they will contain the existing values of these fields.

Below this is a group of fields labelled "Tags". Grouping fields together in this way helps to improve the organisation of the page and make it less cluttered. The left hand side of this section contains a list of the tags currently attached to the meal. Items in this list can be selected, and removed by clicking the Remove selected tag button. On the right two options for adding tags to the meal are offered: a combo box allowing the user to select a tag previously used on another meal, or a text box allowing the user to enter a new tag. The list of tags previously used is obtained upon opening the combo box, by way of an API call.

Finally, a reset button and a submission button are placed at the bottom of the form. They are placed here since it is logical for the user to follow the form from top to bottom, encountering first the fields and then the submission button. A link back to the meal browser view is placed at the top right corner of the page.

Incoming navigation The meal creation view can be accessed via a button on the meal browser view, or a shortcut on the homepage. The meal editor view can be accessed by pressing the "edit" action button (the one labelled with a pencil) next to the entry for the meal in the meal browser.

Outgoing navigation The meal creation and meal editor views contain links back to the homepage and the list of meals, and a button to submit the form.

Mockup A mockup of this view is shown in Figure 2.8 (p. 41).

2.7.5 Browse meal plans

Description This view will allow the user to browse the list of meal plans stored in the database, including searching, sorting and deleting facilities. It will also have a hyperlink to the "create meal plan" view, as well as "edit meal plan" views for each meal plan.

Features The meal plan browser view primarily consists of a calendar-like widget. It displays a number of coloured regions, with breaks in the border signifying regions that span multiple rows. Each region represents the days covered by a meal plan, and clicking on the region takes the user to the corresponding meal plan viewer page.

The calendar shows one month of days, plus a few days either side to fill it up to a whole number of weeks. The month displayed is shown above the calendar, and can be changed by clicking the arrow buttons at the top left and right corners of the calendar.

A button leading to the meal plan creation page is placed at the top right corner of the page. It is placed at the top of the page with a large margin around it so that it is easily noticable to a user searching for it.

Incoming navigation The meal plan browser view can be accessed from the home page. The view, create and edit meal plan pages will also have links back to this page.

Outgoing navigation The meal plan browser view links back to the homepage and forward to the create meal plan view. It also contains a link to a view meal plan page for each meal plan listed.

Mockup A mockup of this view is shown in Figure 2.9 (p. 42).

2.7.6 View meal plan

Description This view will display the details of a meal plan to the user, and also contain a delete button and a hyperlink to the associated "edit meal plan" view. It should be suitable for printing.

Features This meal plan viewer displays the meals assigned to each day in the meal plan in a list on the left. Days not assigned a meal are left blank (this may be changed to use a hyphen to indicate lack of a meal). The notes associated with the meal are displayed below this, and the two parts are separated by a horizontal rule. The notes are shown in a slightly smaller font, since the focus of the page is on the table above it.

On the right-hand side of the page opposite the list of days are two buttons. The first leads to the corresponding meal plan editor view for this meal plan, and the second asks the user to confirm deletion of the meal plan. At the top right corner of the page is a hyperlink back to the meal plan browser for easy navigation.

Incoming navigation The meal plan viewer can be accessed by selecting a meal plan from the meal plan browser. The edit meal plan view also links back to this view.

Outgoing navigation The meal plan viewer has links back to the homepage and the meal plan browser, as well as a link to the meal plan editing view.

Mockup A mockup of this view is shown in Figure 2.10 (p. 43).

2.7.7 Create meal plan

Description This view is the first of two steps in meal plan creation (the second being handled by the meal plan editor view) and will allow the user to create a meal plan by specifying the start and end dates of the meal plan.

Features This view contains two labelled fields, for the start and end dates of the meal plan. The fields incorporate date-picker widgets, which are activated by clicking the calendar icon to the right of each field. The date-picker widget aids in the selection of a date, allowing the user to focus on their goal rather than having to work out the format of a text-based date field.

Below these fields are two buttons leading to the meal plan editor. The first button allows the user to add meals to the initially empty meal plan, while the second automatically generates a meal plan and allows the user to amend the result. A hyperlink leading back to the meal plan browser is placed at the top right hand corner of the page.

Incoming navigation The meal plan creation view can be accessed via a button on the meal browser view, or a shortcut on the homepage.

Outgoing navigation The meal plan creation view links to the homepage and meal plan browser.

Mockup A mockup of this view is shown in Figure 2.11 (p. 44).

2.7.8 Edit meal plan

Description This view will allow the user to amend an existing meal plan using the meal plan editor.

Features The meal plan editor view contains a list of the days covered by the meal plan, and the meal assigned to each one (or the placeholder text "Click to change"). Clicking on the name of the meal opens the dialogue box allowing a meal to be selected. A "delete" button is placed on each row to allow clearing the row to the "no meal chosen" state, by way of a background API call.

The contents of the dialogue box are similar to that of the meal browser view (see Section 2.7.5 (p. 36)), but replacing the tags column with an indication of how much the meal is recommended (the score) using a 5-star system. The meal suggestions are generated upon opening the dialogue box by way of a background API call, to avoid leaving the page. Clicking on the name of a meal in this list adds/updates the serving in the database, also by performing a background API call.

Below the list of days is a large text box that the user can enter associated notes into. The submission and reset buttons are also placed here; however since all modifications are done via API calls, the submission button simply directs the user straight to the meal plan viewer. A link back to the meal plan viewer is located at the top right-hand corner.

Incoming navigation The meal plan editor view can be accessed by pressing the "edit" button in the view meal plan page.

Outgoing navigation The meal plan creation view links to the homepage, meal plan browser and meal plan viewer.

Mockup A mockup of this view is shown in Figure 2.12 (p. 45).

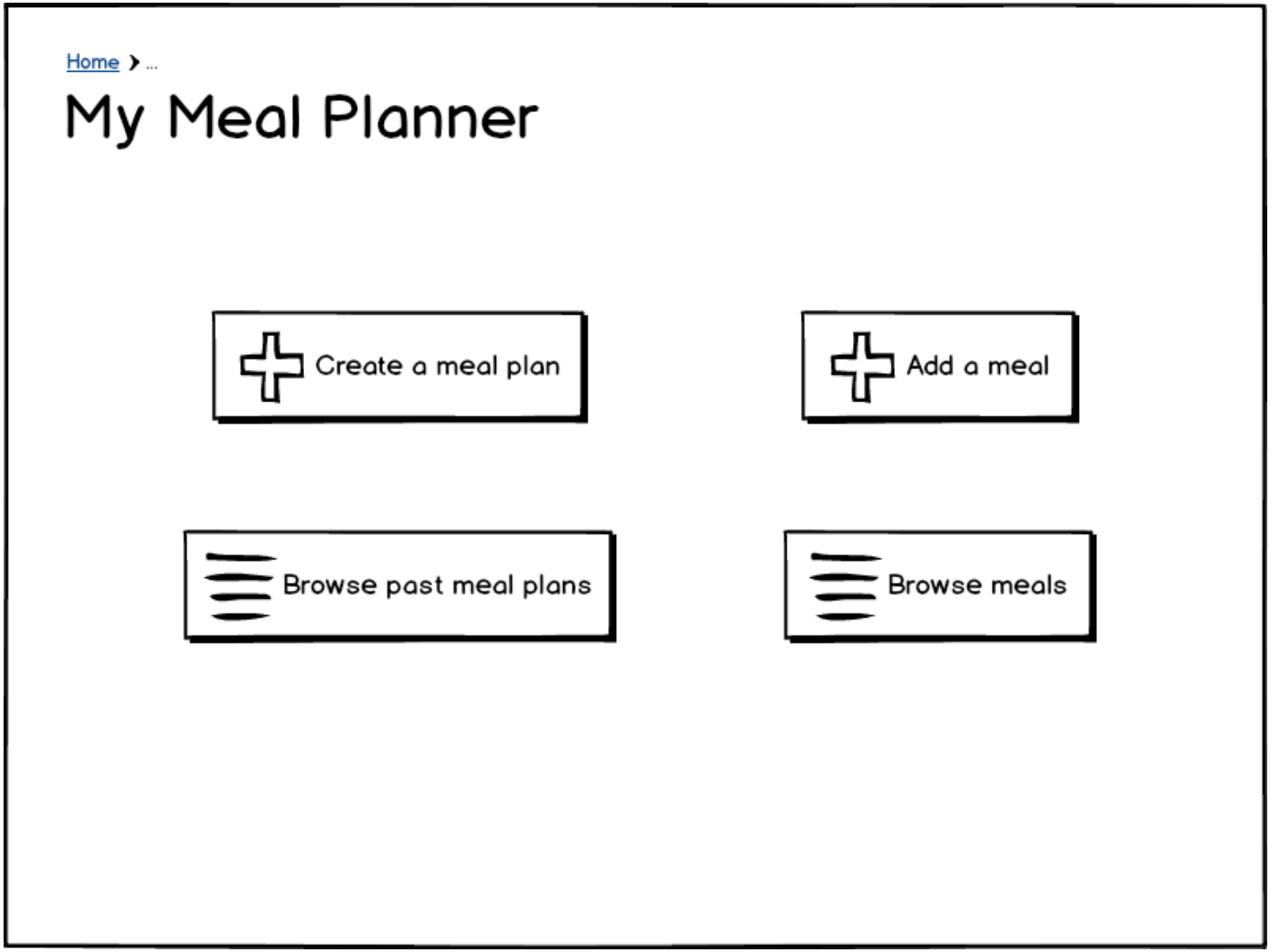

Figure 2.6 – GUI mockup of home page

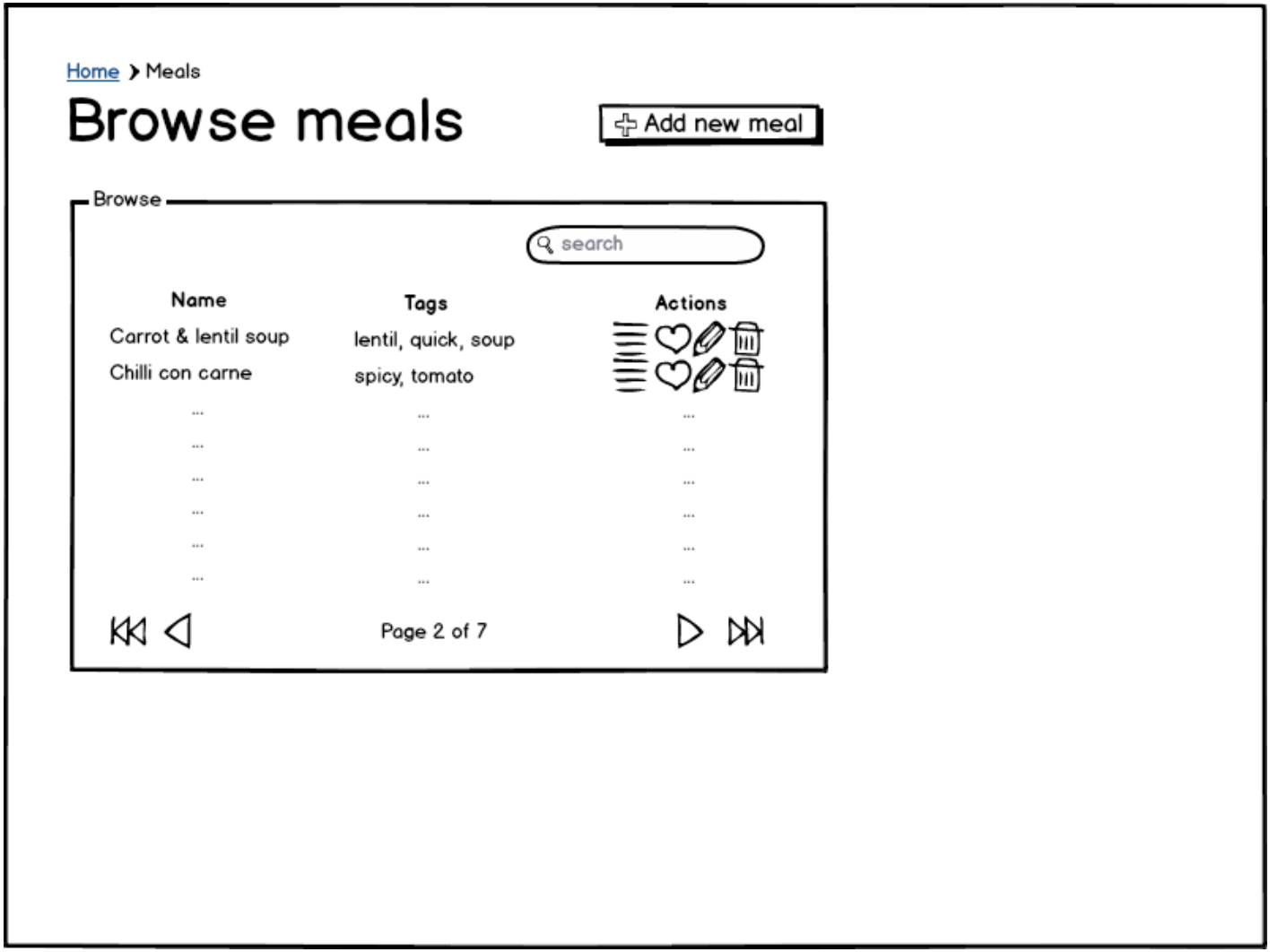

Figure 2.7 – GUI mockup of "browse meals" view

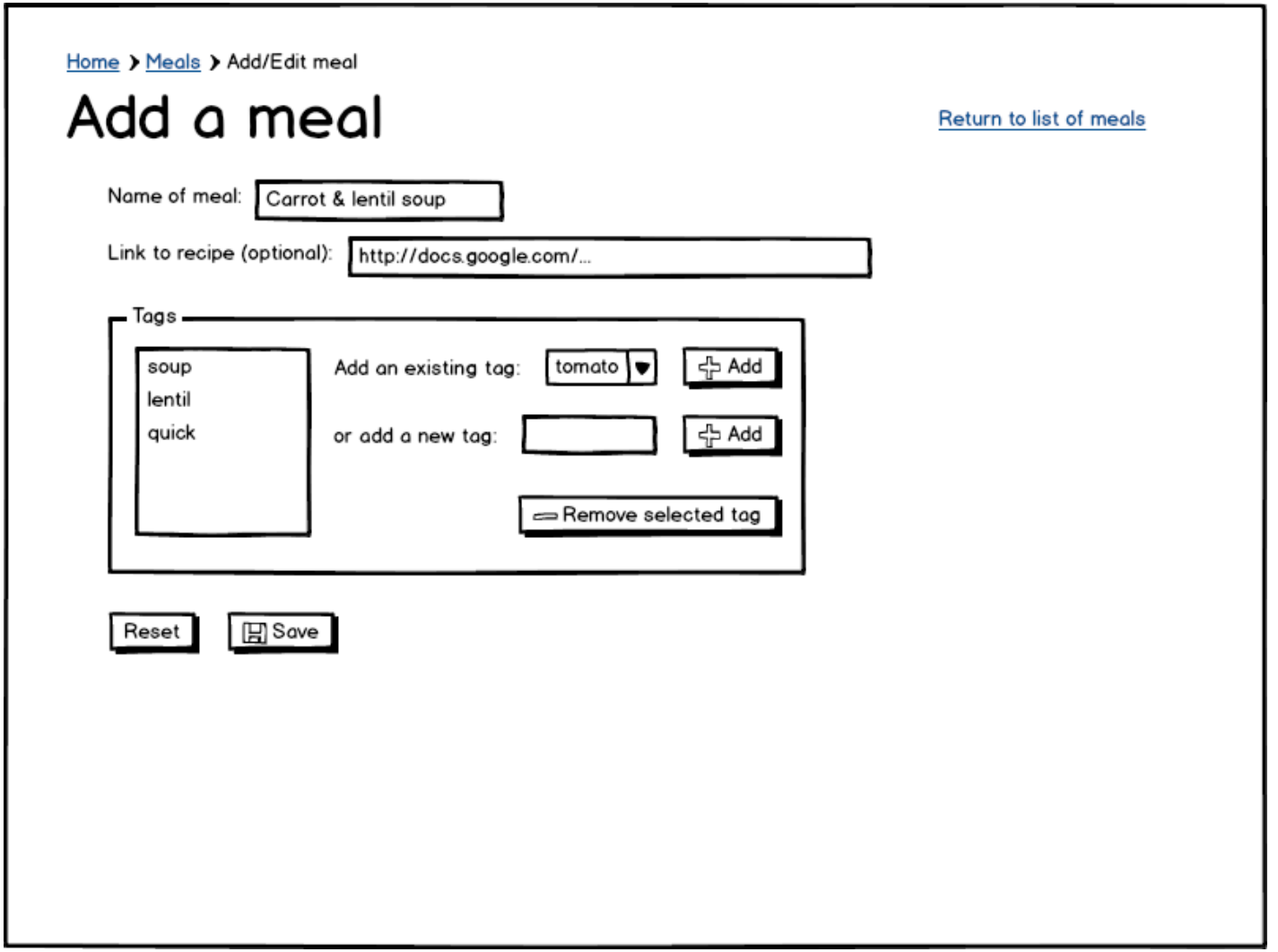

Figure 2.8 – GUI mockup of "add/edit meal" view

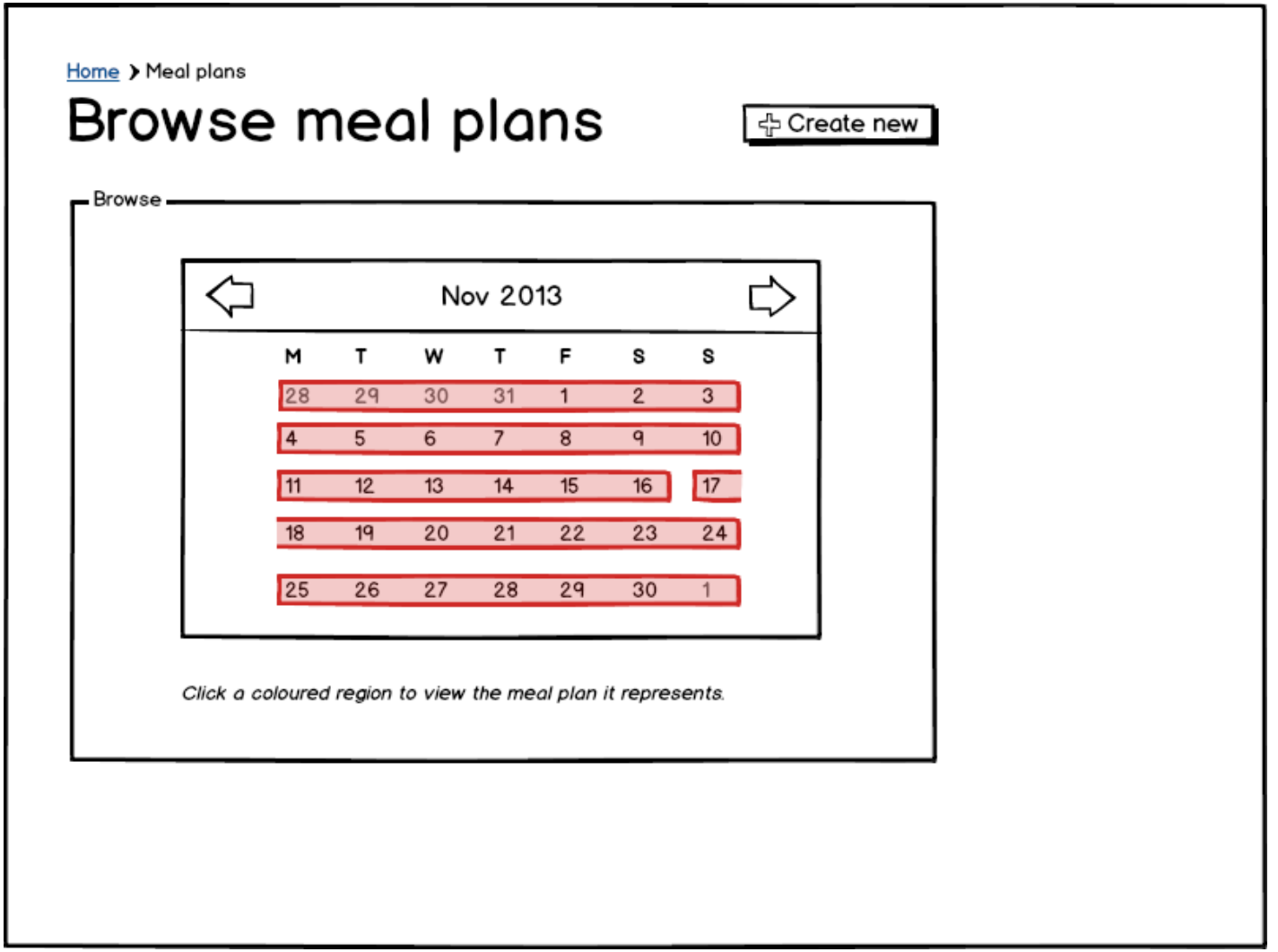

Figure 2.9 – GUI mockup of "browse meal plans" view

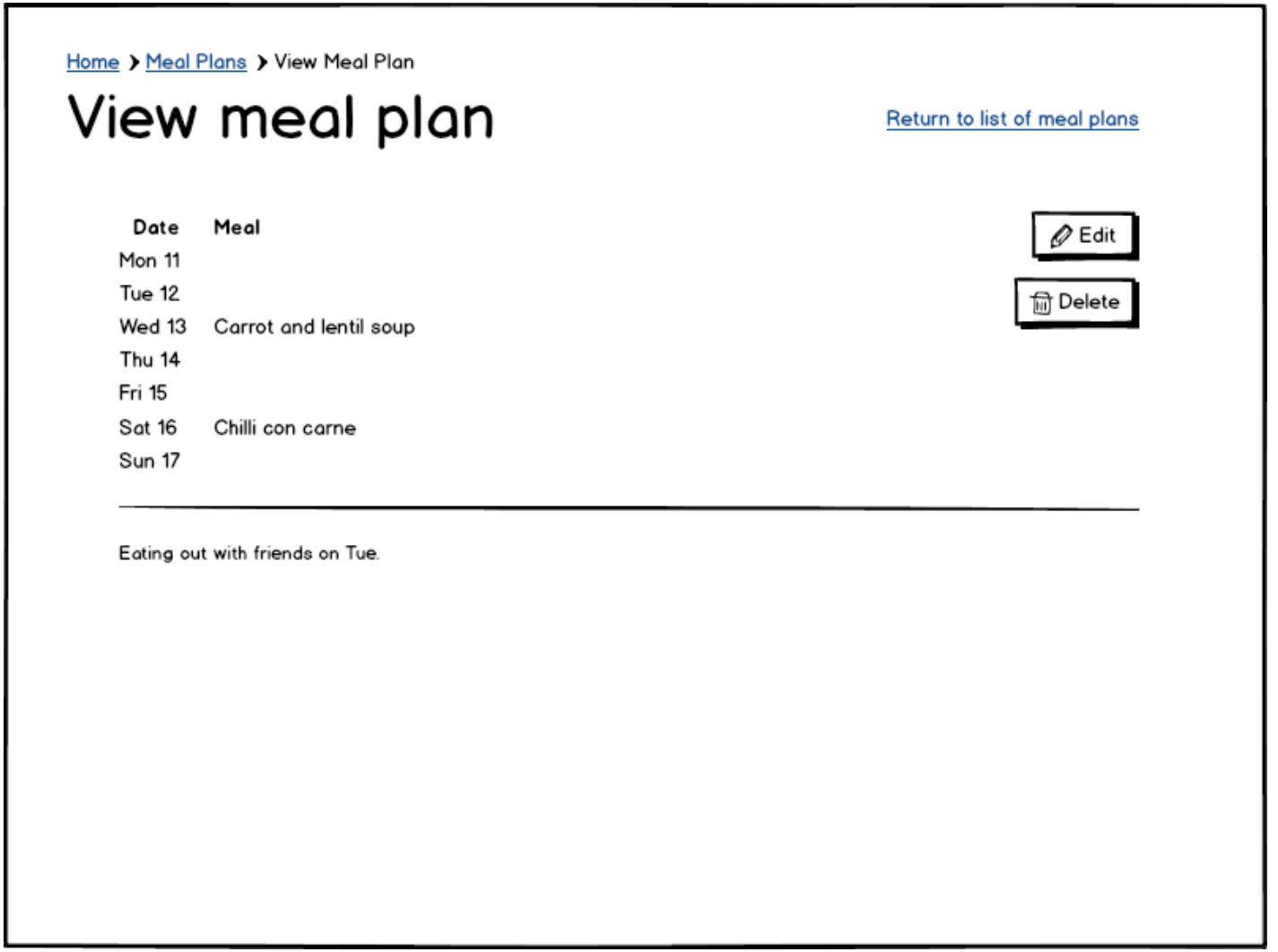

Figure 2.10 – GUI mockup of "view meal plan" view

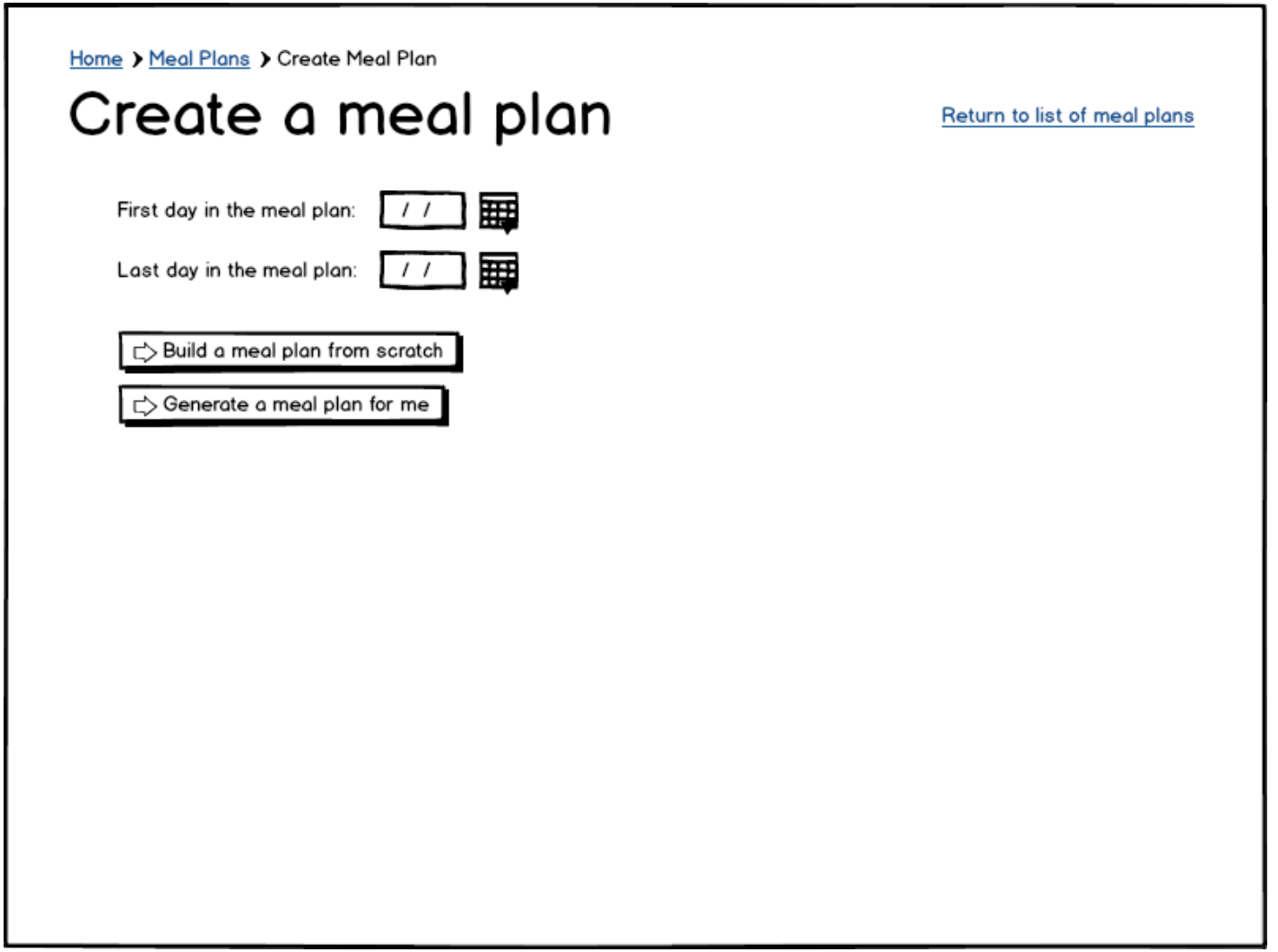

Figure 2.11 – GUI mockup of “create meal plan” view

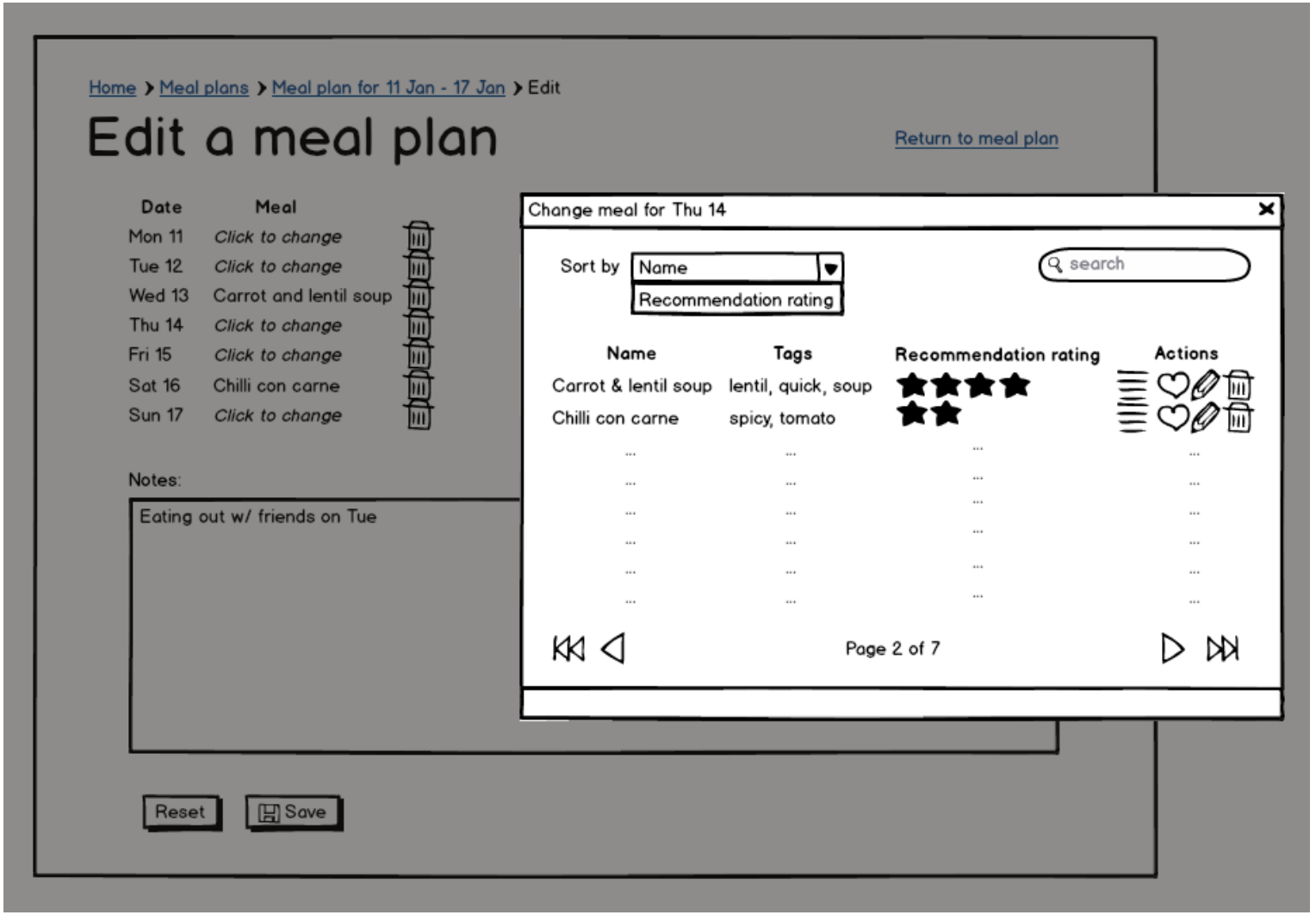

Figure 2.12 – GUI mockup of "edit meal plan" view

2.8 Test strategy

The system will need to be tested in various ways before it is ready for use, to eliminate as many faults as possible. This is important as if an unexpected error does occur then the application is unlikely to behave as the user intended.

One way in which it will be tested is implicit black-box testing by both the developer and the user. The developer will test the code as it is written, ensuring that they perform the intended task in general. Upon delivery of the first release of the software to her, the user may decide to explore the application and verify that it meets the requirements. This is primarily a top-down method of testing, as it checks whether the program works as a whole without considering the individual steps it takes.

While this method of testing will find some of the larger and/or more common flaws in the application, it may take a long time to find subtle bugs that may only become evident after weeks or months of use. For this reason, white-box testing will be performed on the application by the developer; every possible function that the application should be capable of will be checked to ensure it works correctly. An example of the testing that will be perfomed in this category is to check that form validation (both on the client and the server) responds correctly to valid, borderline and erroneous data. This testing method is bottom-up; it mostly focuses on checking the validity of small parts of the program, but less on the integration of the the parts. Combining black-box (top-down) and white-box (bottom-up) testing produces a wide variety of scenarios, essential for effective testing of a system.

The testing of the application will be documented, but it need not be to a huge level of detail. The black-box testing performed by the developer will take place as each part of the program is written, and the client would not want to be overwhelmed with a huge list of specific features to test, so this testing could be reported by simply recording whether each web page view works as intended. The white-box testing is more complex and so will be logged to a greater degree of detail, listing each feature being tested, the input data entered and the expected and actual result. A short description of the results of the testing will also be written to summarise the outcomes.

A general plan for the testing of the system is given in Table 2.3.

Test group	Description
1	Test that all UI buttons and hyperlinks direct the user to the correct destination.
2	Test that all UI non-hyperlink actions produce the correct effect.
3	Test that all form validation is correct.
4	Test that all server-side flow control is correctly performed.

Table 2.3 – General test plan

2.9 Security and integrity of data

Keeping the system secure is a major issue in any application. The machine running the server software will be placed behind an Internet firewall in order to reduce the risk of unauthorised access to the system by an attacker. However, since there is no user account system currently required in the application, nor any storage of personal data, there is no issue with unauthorised access to one user's data by another.

It is possible that in a future version of the system, a user account system will be needed. If this is the case, then a secure password system such as bcrypt[7] would be used. A password hashing system like this does not store the password in a way in which the original password text could be obtained, and deliberately uses a lengthy hashing algorithm in order to reduce the rate at which a brute-force attack could take place. This is a standard practice used in many applications across the web.

The data must also be protected from corruption. Validation at both the UI and database level will ensure that only correctly formatted data is stored in the application. Backups of the database will be taken regularly and stored in multiple locations in order to reduce the damage caused if data loss occurs.

[7] See `http://en.wikipedia.org/wiki/Bcrypt`

3. Implementation and Testing

The source code of the system is given in Appendix F (p. 141). The user manual can be found in Appendix D (p. 93) and the system maintenance manual in Appendix E (p. 110).

3.1 Major changes from design made in implementation

It is inevitable when developing a system that the final implementation may be slightly different to that outlined in the design. This section will enumerate some of these differences and explain how they arised.

- In the implementation, the `meal` table contained an extra field compared to the table definition given in Section 2.4 (p. 24) of the Design chapter. This field is named `searchtext` and contains no data that is not stored elsewhere in the `meal` or `tag` (therefore, the database is technically not in Third Normal Form). The purpose of this field is simply to accelerate searching the list of meals, by caching all pieces of text that may be searched (meal name, recipe URL and tags) in one string.
- The scoring algorithm was improved for the implementation; code was added that allowed the presence of the same tags as the meal being scored on meals nearby to deduct from the score. This was done after it was realised that the algorithm given in Section 2.5.1 (p. 26).

3.2 Testing

A strategy for the testing of the system was given earlier in Section 2.8 (p. 46). Based on this, a detailed test plan was created to thoroughly test each layer of the application. This is listed in Appendix B (p. 63).

The results of this testing are listed in Appendix C (p. 78); it can be seen that out of 161 tests, 144 of these succeeded, the remaining 17 having failed. The tests that failed are discussed below.

3.2.1 Failed tests

Tests 1.3.3 and 1.4.3: "Return to list of meals" button missing in meal creation/editing form

Test group description Test that the meal creation/editing form hyperlinks and buttons direct the user to the appropriate pages.

Input Click "Return to list of meals".

Expected result Directed to meal browser.

Actual result Button is not present on form. See Figure C.1 (p. 86).

Reason This button was not added since there is already a hyperlink back to the meal list in the top navigation bar.

Test 1.7.3: "Return to list of meal plans" button missing in meal plan creation form

Test group description Test that the meal plan creation form hyperlinks and buttons direct the user to the appropriate pages.

Input Click "Return to list of meal plans".

Expected result Directed to meal plan browser.

Actual result Button is not present on form. See Figure C.2 (p. 87).

Reason This button was not added since there is already a hyperlink back to the meal plan list in the top navigation bar.

Test 1.8.5: "Return to list of meal plans" button missing in meal plan editor

Test group description Test that the meal plan editor hyperlinks and buttons direct the user to the appropriate pages.

Input Click "Save".

Expected result Directed to meal plan viewer for meal plan being edited.

Actual result Button is not present on form. See Figure C.3 (p. 88).

Reason This button was not added since there is already a hyperlink back to the meal plan list in the top navigation bar.

Test 2.11.6: "Reset" button missing in meal plan editor

Test group description Test that the editing of a meal plan functions correctly.

Input Assign two new arbitrary servings to the meal plan, and remove two other servings from the meal plan, before pressing the "Reset" button.

Expected result The list of servings is reverted to the state it was in when the page was loaded. This change is reflected in the database.'

Actual result No "Reset" button on form.

Reason A "Reset" button for this form was not implemented since it was considerable more complex than other forms (it requires the database to be modified rather than just resetting the contents of fields in the browser). Time constraints were also an issue.

Test groups 3.3 and 3.4: Incorrect date validation

Out of the 24 tests in groups 3.3 and 3.4, 10 of these failed whilst only 14 passed. This low success rate was due to the poor validation code on both the client and server. To fix this issue, the following could be done:

- Add range validation to the day and month parts, to ensure they are not greater than their maximum value.
- Prevent out-of-range values from being "normalised" to the next month (e.g. "00/11/2013" is converted to "31/10/2013").
- Return a more detailed error message than "Bad Request" if server-side validation fails.

Tests 4.14.1 and 4.14.2: "Reset" button missing in meal plan editor

Test group description Test that resetting a meal plan functions correctly.

Input Add/delete an arbitrary serving from a meal plan, then press the "Reset" button.

Expected result The meal plan is reverted to the state it was in before the change.

Actual result No "Reset" button present on form.

Reason See the reasoning behind test 2.11.6 above.

3.2.2 Conclusions

It can be seen from the testing that there are a number of issues with the application. These issues can be summarised as follows:

- Some buttons were not implemented despite being present in the HCI design, due to there already being a hyperlink to the destination page present in the navigation bar.
- The meal plan editor "Reset" button was not implemented due to time constraints.
- The date validation on both the client and the server has room for improvement.

4. Appraisal

4.1 Feedback from user with analysis

This section lists both praise and constructive criticism that my user has made about the application. Some analysis of each piece of feedback is also given.

4.1.1 Mid-implementation feedback

About half of the way through the implementation, I prepared a version of the application as it stood and set it up for my user to explore. In return, she provided some feedback on the application which I was then able to act upon, including:

- *"When editing a previously inputted meal, tags which are visible on the list of meals page are not showing up. Are they being saved?"*

  My user discovered a bug which prevented the tags of a meal from being shown in the meal editing page (even though they were stored in the database). This was shortly amended in response to the feedback.

- *"When adding tags for the first time, they do not save for later use."*

  It was also found that the drop-down box of existing tags was not correctly populated when the page was loaded. Because of this and the previous point it could easily be mistaken that tags that were added to meals during creation were not being saved. Again, this issue has since been fixed.

- *"When adding a tag, if enter is pressed instead of the save tag button, a null tag is formed. Seems natural to press enter?"*

  My user commented that it would be expected that pressing Enter in the "new tag" field would add the tag; instead it attempted to either submit the form or add an invalid tag. Functionality was added to add a tag when Enter is pressed.

Receiving feedback during the implementation phase was very helpful in getting an idea for how well the project was coming along, and for discovering issues that I had not seen myself.

4.1.2 Final feedback

Once the implementation, testing and documentation was complete, I demonstrated the complete application to my user. She provided a number of positive comments as well as areas of improvement. The positive feedback included:

- *"The program works very well and I find it useful. It makes planning and shopping for meals simpler and easier."*

- *"I like the way that it is intuitive and learns preferences from previous meal plans."*
- *"It's good to have all my recipes and meal ideas together in one place."*
- *"The 'favouriting' feature is an additional bonus. Meals that everybody enjoys are suggested more frequently."*

Some constructive criticism that was given by my user was:

- *"Ideally, it would be more useful if there was a print button where a meal plan can be printed out."*

  My user feels that a dedicated "Print" button, if added to the meal plan viewer, would improve the user experience of the application.
- *"I was slightly confused by the shifting of the meal list buttons when there's no recipe attached."*

  The "action buttons" in the rightmost column in the meal list are not in fixed columns and so shift towards the left if the first button (the recipe button) is hidden. This can be confusing since the other three buttons are in different places for meals with and without recipe URLs assigned to them. An possible amendment for this is given as part of the improvements listed in Section 4.3 (p. 55).
- *"It would seem more natural to have a page navigation bar at the bottom of the page."*

  In the meal list view, the bar containing the page navigation buttons and the page number display is only present at the top of the list. The computer I mainly tested the application on has a fairly high screen resolution, so scrolling the screen was not much of an issue. In comparison, my user's screen is a lower resolution and so having to scroll all the way to the top of the page simply to go to the next page became somewhat distracting. It was determined that the addition of a copy of the page bar at the bottom of the meal list would greatly reduce the time wasted on navigating between pages.

4.1.3 Authentication

These quotes accurately represent the views and opinions of the client.

Signature of client: .....

4.2 Comparison with original objectives

The project objectives listed in Section 1.11 (p. 16) of the analysis are discussed here in relation to the finished project.

The system must:

1. **allow the user to fully manage a collection of meals, and generate suggestions based on other servings of meals.**

   Both my user and I agree that the application does fulfil the general purpose set out by this objective. Meals can be added, edited and removed, and suggestions for servings can be generated. Suggestion choosing is influenced by the servings of other meals.

2. **allow the user to add a meal to the database, supplying a name, hyperlink to a recipe and one or more tags.**

   The application has working functionality for adding a meal to the database. Information that can be added to the meal includes its name, a hyperlink to a recipe, whether or not it is marked as a favourite and a number of tags. Therefore, I believe this objective is fully satisfied.

3. **allow the user to update the same three pieces of information as in objective 2 stored about a meal.**

   Functionality also exists to edit the appropriate information about meals stored in the database. Again, this objective is satisfied.

4. **allow up to 100 meals to be stored in the database, with room for expansion.**

   The database management system used for the application is powerful enough to be able to efficiently store and query millions of records, which greatly out-numbers the amount specified in this objective. However, it is untested whether the application's performance during searching and suggestion generation will still be reasonable at these volumes of data. Despite this, the objective is considered to be fulfilled.

5. **allow a meal to be marked as favourited or not.**

   My user commented that the "favourite" status of a meal can be seen and updated easily in the meal browsing view, so the objective is achieved.

6. **allow the user to remove meals from the database.**

   This objective is attained since meals can be removed from the database via the meal list browser.

7. **allow the user to search for meals by name or by tag. The search operation must complete in under 0.1 seconds for a database of 50 meals, excluding network transmission delays. The view should be paginated.**

   Meals can be searched for by any text matching the meal's name, recipe URL or tags. There is no noticeable delay in searching a database of around 45 meals other than that attributed to network transmission, although the search function has not been specifically timed. The results view (which is simply part of the meal list browser) has a working paging mechanism. Because of this I consider this objective to be more or less fulfilled despite the lack of detailed timing information.

8. **allow the user to browse all meals in the database. The view should be paginated, and the user must be able to sort the results by the default order (order of primary key), by alphabetical order of name or by "score" (a numerical quantity calculated based on the suggestion criteria).**

   The user can browse all meals in the database in a paged view, but the ability to sort the list was not implemented in the released version of the application. Meals listed when choosing a suggestion are always sorted by score whilst those in the meal browser are sorted by name. As a result, this objective is only partially accomplished. Despite this, my user

has responded that the lack of ability to sort the meal list is not a major issue and will not greatly affect her usage of the application. This flaw is discussed further as part of the improvements given in Section 4.3 (p. 55).

9. **allow the user to browse all meals in the database that have a specific tag. The two additional criteria in objective 8 also apply here.**

   This activity can be done by simply searching for the tag in the meal browser, resulting in a paged but unsortable list. Again, this objective is only partly attained.

10. **be able to automatically create a meal plan, which is stored in the database. The meal plan generation should take no more than 0.5 seconds.**

    The application allows meal plans to be created with automatically populated servings. However, although detailed timing information has not been obtained, it can be seen that the process takes longer than 0.5 seconds to take place. Usually, about 3 seconds pass before the result page is displayed; even when compensating for network and request-processing delays, it is clear that it is greatly exceeding the given time, so I consider this objective to be incomplete. Despite this, I believe that in hindsight the time limit of 0.5 seconds is too low for the complexity of the resulting algorithm.

11. **allow the user to manually create a meal plan, assisted by the database views described in objectives 7, 8 and 9. This meal plan must be able to be saved in the database.**

    This objective is somewhat satisfied, since a user can create a meal plan using the list of meals. Some of the features in the objectives referred to by this one are not implemented for the creation of meal plans (such as searching and sorting), but my user has commented that she does not find them vital to the meal plan creation process.

12. **allow up to 100 meal plans to be stored in the database, with room for expansion.**

    Likewise to objective 4, the database can store a very large number of records and so this objectively is definitely achieved.

13. **allow the user to browse the past meal plans stored in the database. The view should be paginated.**

    Users are able to browse meal plans in a paged calendar-like view. This objective is fully accomplished.

14. **allow the user to update the meal plans stored in the database. The updatable information should include the meal servings associated with the meal plan as well as the start and end dates.**

    This objective is mostly fulfilled, as the servings associated with meal plans can be edited easily. However, there is currently no way to change the start or end date of a meal plan without deleting and re-creating it; because of this, only part of this objective was accomplished. This flaw is elaborated upon as part of the improvements given in Section 4.3 (p. 55).

15. **ask the user to confirm when changing the start or end date of a meal plan would leave meal servings outside of the range. If the user does confirm, these invalidated meal serving records must be deleted.**

    This objective is not satisfied at all, due to the lack of facilities for changing the start and end dates of meal plans. See the response to objective 14 for more information.

16. **allow the user to remove meal plans from the database.**

    Meal plans can be removed (with confirmation) from the list, so this objective has been completed.

17. **allow the user to print meal plans stored in the database.**

    My user has commented that whilst it is possible to print meal plans, the printed page could be more aesthetically pleasing and the addition of a "Print" button would improve the experience. Therefore, this objective is technically achieved but not to a high standard.

18. **provide web page views suitable for mobile devices as well as those suitable for desktop computers.**

    My user and I have found no flaws or inconveniences with the mobile-optimised web pages, compared to the desktop pages. Because of this I believe that this objective has been accomplished.

4.3 Future improvements

There are a number of improvements that could be made to the application as it stands; some of these are features that were planned to be implemented but were not due to time constraints, and others are features that were not planned at all but could be worthwhile additions to the application.

Features that could be added to the application include:

4.3.1 Major improvements: missing features and major bugs

A dedicated mobile-optimised version of the website. Many modern web applications provide two sets of web pages with (almost) identical features. One is designed from the bottom-up for desktop browsers and the other for mobile devices. The Meal Planner application only provides one site, which uses stylesheets included with the Bootstrap framework to render the content differently on desktop and mobile. Whilst this makes developing the site easier, as the web pages grow in complexity it becomes very difficult to produce the desired look on both platforms using the mostly the same HTML and CSS code. Supporting a separate, mobile-optimised site will allow the pages to be designed to better suit devices with touchscreens and smaller displays.

The ability to sort the list of meals. This feature was originally planned to be implemented and is in fact mentioned in objective 8. It was not implemented due to time constraints.

The ability to search suggestions in the meal plan editor. Currently the meal search functionality only works in the meal browser, and not in the meal plan editor. This is mainly due to the fact that implementing searching would require either the suggestions to be re-generated every time the search query is changed (which is relatively slow) or the suggestions to be saved on the server (which raises questions about where to store them, how long to save them for and how to deal with the user having more than one instance of the meal plan editor open at a

time). The complexity of this issue led to it not being implemented in time for the release of the application; however the non-functioning search box is still visible in the meal plan editor.

The ability to edit the start/end date of a meal plan. In the current version of the application there is no way to change the bounds of a meal plan without deleting and re-creating it, forcing the user to have to re-enter all servings attached to it. The ability to edit the meal plan bounds is part of objective 14, but was not implemented because of time constraints.

Correctly functioning date validation. As illustrated by the testing of the system, it can be seen that the current validation of meal plan start and end dates is faulty and must be improved. This could be rectified by using an existing, well-tested third party library for parsing dates in Javascript that the basic string processing that is currently done on the client.

4.3.2 Minor improvements: minor bugs and "nice-to-haves"

Print button and a more well-designed printable version of meal plan pages. Currently, there is no dedicated printable version of the meal plan viewing page, so the printed page contains buttons, navigation hyperlinks and the footer, which are unnecessary and visually distracting from the main body of the page. This could be improved by hiding certain elements when the browser prepares to print the page.

Additionally, my user has explained that adding a "Print" button to the page itself would make it easier to print the page (rather than having to find the button in the browser's menus).

Styling to stop a hidden recipe button from moving the other action buttons. When the recipe button in a row in the meal list is hidden, the other three action buttons move to the left due to the left-justification of the column. This could be amended by inserting a spacer element (one which takes up space on the page but has no content) instead when the recipe button is hidden.

Code to detect and warn the user about creating a meal plan that overlaps with another. This was considered as a feature that could be added to the application. It was not given much priority due to there existing circumstances under which overlapping meal plans may be desired by the user.

Buttons in the meal plan creation form to allow date ranges to be automatically filled in. The addition of buttons that could automatically set the start and end dates based on date ranges such as "this week" and "next week" was considered. However, this was not a major issue and so was not implemented in time for the release.

Ignoring the existing serving when a meal plan serving is replaced. If a meal is already present in a serving slot when that slot is edited in the meal plan editor, the serving of that meal is currently included in the scoring. Because this serving appears to be on the same date as

the day being edited, this meal ends up being given a very low score (due to its "closest serving distance" being zero). This could be improved by making the scoring algorithm exclude the serving with the same date and meal plan identifier as the serving being edited.

Adding a "clear" button to the meal search box. The addition of a button to simply clear the contents of the search box on the meal list could be a useful addition to the application. Its low priority and the fact that some browsers (mostly mobile ones) insert a button to do this anyway led to it not being implemented.

Validation of URLs. This was originally going to be implemented in the application, and over 40 tests of valid and invalid edge-case URLs were proposed. Due to the relatively priority of this feature it was not implemented and the tests were removed from the plan.

Appendices

A. Transcript of interview with user dated 9 October 2013

Date:	9 October 2013
Time:	19:45
Length of recording:	12 min 3 sec
Participants:	Kier Davis (K)
	Stephanie Davis (S)

The mark *[...]* indicates speech that could not be transcribed.

(K) So, how do you get ideas for meals to cook?

(S) From my head.

(K) Do you consult anyone else when making these choices?

(S) Yes, I ask my family, but they're not very good at giving me feedback. None of them really want to spend any time sorting out meals for the week.

(K) Okay, how do you go about putting these ideas together into a menu?

(S) I have to organise the meals so that similar things don't go together. So a rice dinner, like curry, would not be followed by a rice dinner like chilli. They would be interspersed with either potato, pasta or bread style meals, in a rotational style thoughout the week.

(K) Okay. Are there any other criteria you consider when putting this list together?

(S) Yeah, try not to have the same thing two weeks running, so if you have chilli one week, I'd try not to have chilli the next week, but try to have variety.

(K) So you wouldn't have the same meal too often.

(S) Yeah, and also try to find a meal that everyone enjoys.

(K) How do you define whether two meals are similar or not, and should not be served too close together?

(S) Right, well it's not just as simple as whether or not its like a rice dish, like chilli and curry both get served with rice, but its not just as simple as that. Tomato-y meals, like a tomato-y, I don't know - I try to have variety. Meals which are not too similar are not served either too often or next to each other in the week. So it might not just be like the thing it's with, it might be the fact that it's mainly tomato-y, say, and the next meal would not be. It might involve vegetables with a white sauce or something.

(K) So, what do you find are the most difficult areas of this whole process?

(S) Inspiration, trying to find things. I don't have a big list of things, therefore every time I come to do it, every once a week, I'm back at square one trying to find things everybody likes.

(K) And do you think that a computerised system to aid this process would benefit you?

(S) Yes, because I'm hoping it would have all the suggestions in there, and saves me having to be inspirational, and think up individiual things.

(K) What are the most important features you would want in a system like this?

(S) I suppose, something that stops me having to be inspirational and *[...]* that will do that bit for me - it'll find the interesting meals, even if I have to put them all in to begin with. It will save me on a weekly basis having to do that. Maybe, have a rating system for meals, so that everybody can rate meals, so that if someone didn't like it perhaps it wouldn't come up, I don't know.

(K) Are there any other features you'd be interested in, maybe not essential, but nice to have?

(S) Maybe the rating thing, that's not essential; the essential thing is it's going to come up with different meals on a rolling programme.

(K) What information would you want stored about each meal in the system?

(S) *[...]* some criteria with which you'd be able to compare each meal, so obviously you want what it's usually served with, the rice, potatoes, pasta aspect, but maybe whether or not the meal is a tomato meal, or a white sauce meal, or a dry meal.

(K) Like categories?

(S) Yeah, maybe, so that you've got a way of knowing whether two meals are similar or not.

(K) What about, somewhere to store a link to the recipe?

(S) Yeah, that'd be handy. Most of my recipes are in, well some of them are bookmarked web pages, but most of them are recipes that I've got on my Google Drive, so I could export those or link you to them or something.

(K) You could copy and paste in the link to it, into the system.

(S) That'd be useful, yes, because I use those a lot.

(K) Would there be any users using it other than you?

(S) Nobody else tends to do it, but there's no reason why they couldn't do it. It's really me who does the food shopping and the food cooking.

(K) Alright, how frequently would you expect to use the system?

(S) Once or twice a week. I tend to plan everything on a Sunday night for the rest of the week and shop on Monday, but then I tend to shop again on Friday as well, so it might be that Monday does Monday to Friday and Friday does the weekend.

(K) Can you give a rough estimate, a ballpark figure, for the number of meals that would need to be stored?

(S) I don't know, 50? Can you add to them? Is there a way to be able to add to them?

(K) Yeah, you'll be able to edit the list and add new ones, remove old ones.

(S) Right, okay, maybe 30 to begin with then.

(K) Okay, where would you physically use the application?

(S) What, what room?

(K) Yeah.

(S) Probably in here in the kitchen, on my computer or on my - can I use it on my phone?

(K) Yes.

(S) Okay.

(K) Would you want to use it outside the house at all, like at the shops?

(S) Probably not, I'd use it to generate a shopping list, whether or not you can integrate something like that as one of the optional features, a way to help me to sort out a shopping list on it. That's definitely what I'd use it for, I decide what we're having and then I use that to work out what shopping I need for Monday. I don't need to do that out the house, I can just do that at home.

(K) Would you want meals that you have flagged as your favourite meals to occur more frequently on the list?

(S) Yes. Oh yeah, another category could be the time that it takes to make it. If something's really fiddly, and it uses up every pan and a whole afternoon's work, I don't really want to do that five days in a week.

(K) Okay.

(S) So there might be a category of whether or not it's an easy meal, a medium meal or a tricky meal.

(K) That's it for the questions, thank you.

The user was also asked to rate the importance of a number of possible application features on a scale from 1 to 5, where 1 is not important and 5 is essential. The results are displayed in Table A.1.

A.1 Authentication

This transcript accurately represents the views and opinions of the client.

Signature of client: .....

Feature	Importance (out of 5)
Adding meals	5
Removing meals	5
Updating meal information	4
Adding pictures to a meal	3
Categorisation of meals (e.g. soup, curry, pasta)	5
Searching for a meal by name	4
Searching for a meal by ingredient	3
Browsing for meals by category	3
Rating meals	3
Viewing nutritional information about meals	2
Viewing when a meal was last served	2
Automatic creation of meal plans	4
Manual creation of meal plans	4
Printing the meal plan	4
Changing your password for the application	3

Table A.1 – Rating of importance of proposed system features by user

B. Plan of testing for system

All tests are given in the following table:

Test group	Description	Test number	Input	Expected result
1.1	Test that the homepage hyperlinks and buttons direct the user to the appropriate pages.	1.1.1	Click “Create a meal plan”.	Directed to meal plan creation form.
		1.1.2	Click “Add a meal”.	Directed to meal creation form.
		1.1.3	Click “Browse meal plans”.	Directed to meal plan browser.
		1.1.4	Click “Browse meals”.	Directed to meal browser.
1.2	Test that the meal browser hyperlinks and buttons direct the user to the appropriate pages.	1.2.1	Click “Home” (above title).	Directed to homepage.
		1.2.2	Click “Add new meal”.	Directed to meal creation form.
		1.2.3	Click recipe button next to any meal.	Directed to recipe linked to meal.
		1.2.4	Click pencil button next to any meal.	Directed to meal editing form for corresponding meal.
1.3	Test that the meal creation form hyperlinks and buttons direct the user to the appropriate pages.	1.3.1	Click “Home” (above title).	Directed to homepage.
		1.3.2	Click “Meals” (above title).	Directed to meal browser.
		1.3.3	Click “Return to list of meals”.	Directed to meal browser.
		1.3.4	Click “Save”.	Directed to meal browser.

1.4	Test that the meal editing form hyperlinks and buttons direct the user to the appropriate pages.	1.4.1	Click “Home” (above title).	Directed to homepage.
		1.4.2	Click “Meals” (above title).	Directed to meal browser.
		1.4.3	Click “Return to list of meals”.	Directed to meal browser.
		1.4.4	Click “Save”.	Directed to meal browser.
1.5	Test that meal plan browser hyperlinks and buttons direct the user to the appropriate pages.	1.5.1	Click “Home” (above title).	Directed to homepage.
		1.5.2	Click “Create new”.	Directed to meal plan creation form.
		1.5.3	Click any coloured region on the calendar view.	Directed to meal plan viewer for corresponding meal plan.
1.6	Test that the meal plan viewer hyperlinks and buttons direct the user to the appropriate pages.	1.6.1	Click “Home” (above title).	Directed to homepage.
		1.6.2	Click “Meal plans” (above title).	Directed to meal plan browser.
		1.6.3	Click “Return to list of meal plans”.	Directed to meal plan browser.
		1.6.4	Click “Edit”.	Directed to meal plan editor for corresponding meal plan.
1.7	Test that the meal plan creation form hyperlinks and buttons direct the user to the appropriate pages.	1.7.1	Click “Home” (above title).	Directed to homepage.

		1.7.2	Click “Meal plans” (above title).	Directed to meal plan browser.
		1.7.3	Click “Return to list of meal plans”.	Directed to meal plan browser.
		1.7.4	Click “Build a meal plan from scratch”.	Directed to meal plan editor.
		1.7.5	Click “Generate a meal plan for me”.	Directed to meal plan editor.
1.8	Test that the meal plan editor hyperlinks and buttons direct the user to the appropriate pages.	1.8.1	Click “Home” (above title).	Directed to homepage.
		1.8.2	Click “Meal plans” (above title).	Directed to meal plan browser.
		1.8.3	Click “Meal plan for...” (above title).	Directed to meal plan viewer for meal plan being edited.
		1.8.4	Click “Return to meal plan”.	Directed to meal plan viewer for meal plan being edited.
		1.8.5	Click “Save”.	Directed to meal plan viewer for meal plan being edited.
		1.8.6	Click recipe button next to any meal.	Directed to recipe linked to meal.
		1.8.7	Click edit button next to any meal.	Directed to meal editing form for corresponding meal.
2.1	Test that searching the list of meals in the meal browser and meal plan editor functions correctly.	2.1.1	Type “carrot” into search field.	Only meals with “carrot” contained in the name or tags are shown.
2.2	Test that pagination of the list of meals in the meal browser and meal plan editor functions correctly.	2.2.1	On any page except the last, click the single right arrow button.	The page number increases by one and the corresponding page of results is shown.

		2.2.2	On the last page, click the single right arrow button.	No effect.
		2.2.3	On any page except the first, click the single left arrow button.	The page number decreases by one and the corresponding page of results is shown.
		2.2.4	On the first page, click the single left arrow button.	No effect.
		2.2.5	On any page except the last, click the double right arrow button.	The page number increases to the maximum and the last page of results is shown.
		2.2.6	On the last page, click the double right arrow button.	No effect.
		2.2.7	On any page except the first, click the double left arrow button.	The page number decreases to one and the first page of results is shown.
		2.2.8	On the first page, click the double left arrow button.	No effect.
2.3	Test that toggling the "favourited" status of a meal in the meal browser and meal plan editor functions correctly.	2.3.1	Click the heart button next to a meal where the heart button is not highlighted.	The heart button becomes highlighted.
		2.3.2	Click the heart button next to a meal where the heart button is highlighed.	The heart button no longer becomes highlighted.
2.4	Test that deleting a meal in the meal browser and meal plan editor functions correctly.	2.4.1	Click the trash can icon next to any meal listed.	A confirmation dialogue box is displayed, allowing the user to choose "Yes" or "No".

		2.4.2	Select “Yes” in the confirmation dialogue box.	The dialogue box is closed and the meal is deleted from the database and removed from the list.
		2.4.3	Select “No” in the confirmation dialogue box.	The dialogue box is closed without the meal being deleted.
2.5	Test that adding a meal to the database functions correctly.	2.5.1	Enter arbitrary but valid meal information into the form, and press “Save”.	A meal with exactly the given information is added to the database.
		2.5.2	Enter arbitrary meal information into the form, and press “Reset”.	All form fields are cleared.
2.6	Test that editing a meal in the database functions correctly.	2.6.1	Alter the contents of the “Name of meal” field, then press “Save”.	The meal is now listed in the meal browser with the altered name.
		2.6.2	Alter the contents of the “Link to recipe” field, then press “Save”.	Pressing the recipe button next to the meal in the meal browser now directs the user to the altered URL.
		2.6.3	Add a new, arbitrary but valid tag to the list of tags, then press “Save”.	The meal is now listed in the meal browser with the new tag present.
		2.6.4	Add an arbitrary, existing tag to the list of tags, then press “Save”.	The meal is now listed in the meal browser with the new tag present.
		2.6.5	Remove an arbitrary tag from the list of tags, then press “Save”.	The meal is now listed in the meal browser with the removed tag omitted.

2.7	Test that the tag list editing interface within the meal creation and meal editing forms functions correctly.	2.7.1	Add the tags “soup”, “lentil” and “quick” to a new meal in an otherwise empty database, then open the drop-down box labelled “Add an existing tag”.	The drop-down box is populated with the tags “soup”, “lentil”, “quick” and no more.
		2.7.2	Add the tags “soup”, “lentil” and “quick” to three seperate meals in an otherwise empty database, then open the drop-down box labelled “Add an existing tag”.	The drop-down box is populated with the tags “soup”, “lentil”, “quick” and no more.
		2.7.3	Select any tag present in the drop-down box and press the “Add” button to the right of it.	The tag appears in the list to the left of the drop-down box.
		2.7.4	Enter any valid tag into the text field labelled “add a new tag”, then press the “Add” button to the right of it.	The tag appears in the list to the left of the drop-down box.
		2.7.5	Select any tag in the list on the left, then press “Remove selected tag”.	The tag is removed from the list.
		2.7.6	When the list on the left is empty, press “Remove selected tag”.	No effect.
2.8	Test that the meal plan browser functions correctly.	2.8.1	Add a single meal plan spanning 3 Nov 2013 to 9 Nov 2013 to an otherwise empty database, then view the meal plan browser.	A single coloured region is shown on the “Nov 2013” page, encompassing the days labelled “3” to “9” inclusive and no further.

		2.8.2	Add a single meal plan spanning 7 Nov 2013 to 11 Nov 2013 to an otherwise empty database, then view the meal plan browser.	Two coloured regions are shown on the "Nov 2013" page. The first encompasses the days labelled "7" to "9", and the second encompasses those labelled "10" and "11".
		2.8.3	Add a single meal plan beginning and ending on 7 Nov 2013 to an otherwise empty database, then view the meal plan browser.	A single coloured region is shown on the "Nov 2013" page, encompassing only the day labelled "7".
		2.8.4	Add meal plans spanning (27 Oct to 2 Nov), (3 Nov to 9 Nov), (10 Nov to 15 Nov), (16 Nov to 23 Nov), (24 Nov to 30 Nov) to an otherwise empty database, then view the meal plan browser.	A layout of coloured regions identical to that shown in Figure 2.9 (p. 42) is displayed.
		2.8.5	On the "Nov 2013" page, press the left arrow button in the header of the calendar.	The "Oct 2013" page is now displayed.
		2.8.6	On the "Nov 2013" page, press the right arrow button in the header of the calendar.	The "Dec 2013" page is now displayed.
		2.8.7	On the "Jan 2013" page, press the left arrow button in the header of the calendar.	The "Dec 2012" page is now displayed.
		2.8.8	On the "Dec 2013" page, press the right arrow button in the header of the calendar.	The "Jan 2014" page is now displayed.
2.9	Test that meal plan deletion functions correctly.	2.9.1	Click the "Delete" button in the meal plan viewer.	A page asking the user to confirm deletion appears, including buttons labelled "Yes" and "No".

		2.9.2	Click the “Yes” button.	The meal plan is deleted from the database and the user is directed to the meal plan browser.
		2.9.3	Click the “No” button.	The user is directed back to the meal plan viewer without the meal plan being deleted.
2.10	Test that meal plan creation functions correctly.	2.10.1	Set the field labelled “First day” to 11 Nov 2013 and the field labelled “Last day” to 16 Nov 2013, then click “Build a meal plan from scratch”.	A new meal plan is created extending from 11 Nov to 16 Nov inclusive, containing no servings.
		2.10.2	With at least 6 meals in the database, set the field labelled “First day” to 11 Nov 2013 and the field labelled “Last day” to 16 Nov 2013, then click “Generate a meal plan for me”.	A new meal plan is created extending from 11 Nov to 16 Nov inclusive, containing 6 servings chosen by the system.
2.11	Test that the editing of a meal plan functions correctly.	2.11.1	Click any of the dates in the list.	A dialogue box containing a list of meals is presented.
		2.11.2	Click the trash can button next to any serving where a meal is set.	The row is restored to say “Click to change” and the serving is removed from the database.
		2.11.3	Click the trash can button next to any serving where a meal is not set.	No effect.
		2.11.4	Click the name of any meal in the dialogue box presented after clicking a date.	The dialogue box is closed and the serving for the clicked date is updated with the same meal.

		2.11.5	Click the close button in the top right hand corner of the dialogue box.	The dialogue box is closd without the serving being modified.
		2.11.6	Assign two new arbitrary servings to the meal plan, and remove two other servings from the meal plan, before pressing the “Reset” button.	The list of servings is reverted to the state it was in when the page was loaded. This change is reflected in the database.
3.1	Test that the “name of meal” field in the meal creation and meal editing forms is correctly validated.	3.1.1	Carrot and lentil soup	Valid
		3.1.2	(empty)	Invalid
		3.1.3	254 repetitions of the character “a”	Valid
		3.1.4	255 repetitions of the character “a”	Valid
		3.1.5	256 repetitions of the character “a”	Invalid
3.2	Test that the “add a new tag” field in the meal creation and meal editing forms is correctly validated.	3.2.1	tomato	Valid
		3.2.2	(empty)	Invalid
		3.2.3	63 repetitions of the character “a”	Valid
		3.2.4	64 repetitions of the character “a”	Valid
		3.2.5	65 repetitions of the character “a”	Invalid
3.3	Test that the “first day in the meal plan” field in the meal plan creation form is correctly validated.	3.3.1	15/11/2013	Valid
		3.3.2	(empty)	Invalid

		3.3.3	2013/11/15	Invalid
		3.3.4	tomorrow	Invalid
		3.3.5	30/11/2013	Valid
		3.3.6	31/11/2013	Invalid
		3.3.7	01/11/2013	Valid
		3.3.8	00/11/2013	Invalid
		3.3.9	15/01/2013	Valid
		3.3.10	15/00/2013	Invalid
		3.3.11	15/12/2013	Valid
		3.3.12	15/13/2013	Invalid
3.4	Test that the "last day in the meal plan" field in the meal plan creation form is correctly validated.	*Same as test 3.3*		
3.5	Test that the last day of the meal plan cannot be before the first day.	3.5.1	First day = 15/11/2013, Last day = 16/11/2013	Valid
		3.5.2	First day = 15/11/2013, Last day = 15/11/2013	Valid
		3.5.3	First day = 15/11/2013, Last day = 14/11/2013	Invalid
		3.5.4	First day = 01/01/2000, Last day = 02/01/2000	Valid
		3.5.5	First day = 01/01/2000, Last day = 01/01/2000	Valid
		3.5.6	First day = 01/01/2000, Last day = 31/12/1999	Invalid

4.1	Test that the meal browser is correctly populated.	4.1.1	Empty the database.	The meal browser displays no meals.
		4.1.2	In an otherwise empty database, create a meal named “Chilli con carne”.	The meal browser displays a single meal with name “Chilli con carne”.
		4.1.3	In an otherwise empty database, create a meal named “Chilli con carne” with recipe URL “http://drive.google.com/”.	The meal browser displays a single meal with name “Chilli con carne” and recipe URL “http://drive.google.com/”.
		4.1.4	In an otherwise empty database, create a meal named “Chilli con carne” with tags “spicy” and “tomato”.	The meal browser displays a single meal with name “Chilli con carne” and tags “spicy” and “tomato”.
		4.1.5	In an otherwise empty database, create meals named “Chilli con carne”, “Carrot and lentil soup” and “Nachos”.	The meal browser displays only the meals “Carrot and lentil soup”, “Chilli con carne” and “Nachos”, in that order.
4.2	Test that the hyperlink to the recipe of a meal is correctly set.	4.2.1	Create a meal with the recipe URL set to “http://drive.google.com/”. Click the recipe button of this meal in the meal browser.	The user is directed to “http://drive.google.com/”.
		4.2.2	Create a meal with the recipe URL set to the empty string.	The recipe button of the meal appears deactivated.
		4.2.3	Create a meal with the recipe URL set to “https://drive.google.com/”. Click the recipe button of this meal in the meal browser.	The user is directed to “https://drive.google.com/”.
4.3	Test that marking a meal as a favourite functions correctly.	4.3.1	Click the heart button next to a meal where the heart button is not highlighted.	The meal becomes marked as a favourite.

		4.3.2	Click the heart button next to a meal where the heart button is highlighted.	The meal no longer becomes marked as a favourite.
4.4	Test that deleting a meal functions correctly.	4.4.1	Click the trash can button next to any meal without any tags and select “Yes” in the confirmation dialogue box.	The meal record is deleted from the database.
		4.4.2	Create a meal with one tag, then delete it.	The meal record and its tag record are both deleted from the database.
		4.4.3	Create a meal with two tags, then delete it.	The meal record and its tag records are all deleted from the database.
4.5	Test that adding a meal functions correctly.	4.5.1	Enter the same information into the fields as is shown in the mockup in Figure 2.8 (p. 41) and press Save.	A meal is created with name “Carrot & lentil soup” and recipe URL “http://docs.google.com/”. Tag records “soup”, “lentil” and “quick” are also created and linked to the meal. These attributes are fully displayed in the meal browser.
4.6	Test that opening the meal plan editor initialises the fields with the correct data.	4.6.1	Click the edit button next to a meal with a known ID.	The ID present in the URL of the destination page is the same as the ID of the meal clicked.
		4.6.2	Create a meal with name “Chilli con carne”, then edit it.	The contents of the “Name of meal” field are “Chilli con carne”.
		4.6.3	Create a meal with recipe URL “http://drive.google.com/”, then edit it.	The contents of the “Link to recipe” field are “http://docs.google.com/”.
		4.6.4	Create a meal with tags “lentil”, “quick” and “soup”, then edit it.	The tag list box displays only the tags “lentil”, “quick” and “soup”.

4.7	Test that editing a meal functions correctly.	4.7.1	Enter the same information into the fields as is shown in the mockup in Figure 2.8 (p. 41) and press Save. Then, edit the meal, changing the name to “Chilli con carne” and press Save.	The name of the meal in the database is changed to “Chilli con carne”.
		4.7.2	Enter the same information into the fields as is shown in the mockup in Figure 2.8 (p. 41) and press Save. Then, edit the meal, changing the recipe URL to “http://drive.google.com/” and press Save.	The recipe URL of the meal in the database is changed to “http://drive.google.com/”.
		4.7.3	Enter the same information into the fields as is shown in the mockup in Figure 2.8 (p. 41) and press Save. Then, edit the meal, adding the tag “tomato” and removing the tag “lentil”.	A “tomato” tag record is created for the meal, and the “lentil” tag record is deleted.
4.8	Test that opening the meal plan viewer displays the correct data for the meal plan.	4.8.1	Create a meal plan with starting date 15 Nov 2013 and ending date 20 Nov 2013.	The page lists days Fri 15, Sat 16, Sun 17, Mon 18, Tue 19 and Wed 20, with no meals or additional notes.
		4.8.2	Create a meal plan with starting date 15 Nov 2013 and ending date 20 Nov 2013, and notes “These are some example notes”.	The page lists days Fri 15, Sat 16, Sun 17, Mon 18, Tue 19 and Wed 20, with no meals. The additional notes read “These are some example notes”.
		4.8.3	Create a meal plan with starting date 15 Nov 2013 and ending date 15 Nov 2013.	The serving list shows only Fri 15.

		4.8.4	Create a meal plan with starting date 15 Nov 2013 and ending date 20 Nov 2013, and add a serving of any meal on the 18th to it.	The page lists days Fri 15, Sat 16, Sun 17, Mon 18, Tue 19 and Wed 20, with the selected meal displayed next to Mon 18.
4.9	Test that editing a meal plan initialises the fields with the correct data.	4.9.1	Create a meal plan with starting date 15 Nov 2013 and ending date 20 Nov 2013.	The page lists days Fri 15, Sat 16, Sun 17, Mon 18, Tue 19 and Wed 20, with no meals or additional notes.
		4.9.2	Create a meal plan with starting date 15 Nov 2013 and ending date 20 Nov 2013, and notes "These are some example notes".	The page lists days Fri 15, Sat 16, Sun 17, Mon 18, Tue 19 and Wed 20, with no meals. The contents of the "Notes" field read "These are some example notes".
		4.9.3	Create a meal plan with starting date 15 Nov 2013 and ending date 15 Nov 2013.	The serving list shows only Fri 15.
		4.9.4	Create a meal plan with starting date 15 Nov 2013 and ending date 20 Nov 2013, and add a serving of any meal on the 18th to it.	The page lists days Fri 15, Sat 16, Sun 17, Mon 18, Tue 19 and Wed 20, with the selected meal displayed next to Mon 18.
4.10	Test that deleting a meal plan functions correctly.	4.10.1	Create a meal plan without any servings, and delete it.	The meal plan record is deleted from the database.
		4.10.2	Create a meal plan record and add one serving of any meal to it.	The meal plan record and the serving record is deleted from the database.
		4.10.3	Create a meal plan record and add two servings of any meals to it.	The meal plan record and the serving records are all deleted from the database.
4.11	Test that creating a meal plan functions correctly.	4.11.1	Create a meal plan with starting date 15 Nov 2013 and ending date 20 Nov 2013.	A meal plan record spanning 15 – 20 Nov 2013 is created.

		4.11.2	Create a meal plan with starting date 15 Nov 2013 and ending date 20 Nov 2013, and notes “These are some example notes”.	A meal plan record spanning 15 – 20 Nov 2013 and with notes “These are some example notes” is created.
		4.11.3	Create a meal plan with starting date 15 Nov 2013 and ending date 15 Nov 2013.	A meal plan record spanning only 15 Nov 2013 is created.
4.12	Test that updating a serving functions correctly.	4.12.1	Create a meal plan with starting date 15 Nov 2013 and ending date 20 Nov 2013. Edit the serving for Mon 18 and select any meal.	A serving record for 18 Nov 2013 and the selected meal is added to the meal plan.
		4.12.2	Following on from the previous test, edit the serving for Mon 18 and select a different meal.	The previously added serving record is deleted and a new one for the new meal is added.
4.13	Test that deleting a serving functions correctly.	4.13.1	Following on from the previous test, delete the serving for Mon 18.	The serving record is deleted from the database.
4.14	Test that resetting a meal plan functions correctly.	4.14.1	Add an arbitrary serving to a meal plan, then press the “Reset” button.	The meal plan is reverted to the state it was in before the change.
		4.14.2	Delete an arbitrary serving from a meal plan, then press the “Reset” button.	The meal plan is reverted to the state it was in before the change.

C. Results of testing for system

The results of the tests listed in Appendix B (p. 63) are given as follows:

Test number	Expected Result	Actual Result	Conclusion
1.1.1	Directed to meal plan creation form.	As expected.	Pass
1.1.2	Directed to meal creation form.	As expected.	Pass
1.1.3	Directed to meal plan browser.	As expected.	Pass
1.1.4	Directed to meal browser.	As expected.	Pass
1.2.1	Directed to homepage.	As expected.	Pass
1.2.2	Directed to meal creation form.	As expected.	Pass
1.2.3	Directed to recipe linked to meal.	As expected.	Pass
1.2.4	Directed to meal editing form for corresponding meal.	As expected.	Pass
1.3.1	Directed to homepage.	As expected.	Pass
1.3.2	Directed to meal browser.	As expected.	Pass
1.3.3	Directed to meal browser.	Button is not present on form. See Figure C.1 (p. 86).	Fail
1.3.4	Directed to meal browser.	As expected.	Pass
1.4.1	Directed to homepage.	As expected.	Pass
1.4.2	Directed to meal browser.	As expected.	Pass
1.4.3	Directed to meal browser.	Button is not present on form. See Figure C.1 (p. 86)	Fail
1.4.4	Directed to meal browser.	As expected.	Pass
1.5.1	Directed to homepage.	As expected.	Pass
1.5.2	Directed to meal plan creation form.	As expected.	Pass
1.5.3	Directed to meal plan viewer for corresponding meal plan.	As expected.	Pass
1.6.1	Directed to homepage.	As expected.	Pass
1.6.2	Directed to meal plan browser.	As expected.	Pass
1.6.3	Directed to meal plan browser.	As expected.	Pass

1.6.4	Directed to meal plan editor for corresponding meal plan.	As expected.	Pass
1.7.1	Directed to homepage.	As expected.	Pass
1.7.2	Directed to meal plan browser.	As expected.	Pass
1.7.3	Directed to meal plan browser.	Button is not present on form. See Figure C.2 (p. 87).	Fail
1.7.4	Directed to meal plan editor.	As expected.	Pass
1.7.5	Directed to meal plan editor.	As expected.	Pass
1.8.1	Directed to homepage.	As expected.	Pass
1.8.2	Directed to meal plan browser.	As expected.	Pass
1.8.3	Directed to meal plan viewer for meal plan being edited.	As expected.	Pass
1.8.4	Directed to meal plan viewer for meal plan being edited.	As expected.	Pass
1.8.5	Directed to meal plan viewer for meal plan being edited.	Button is not present on form. See Figure C.3 (p. 88).	Fail
1.8.6	Directed to recipe linked to meal.	As expected.	Pass
1.8.7	Directed to meal editing form for corresponding meal.	As expected.	Pass
2.1.1	Only meals with “carrot” contained in the name or tags are shown.	As expected.	Pass
2.2.1	The page number increases by one and the corresponding page of results is shown.	As expected.	Pass
2.2.2	No effect.	As expected.	Pass
2.2.3	The page number decreases by one and the corresponding page of results is shown.	As expected.	Pass
2.2.4	No effect.	As expected.	Pass
2.2.5	The page number increases to the maximum and the last page of results is shown.	As expected.	Pass
2.2.6	No effect.	As expected.	Pass
2.2.7	The page number decreases to one and the first page of results is shown.	As expected.	Pass
2.2.8	No effect.	As expected.	Pass
2.3.1	The heart button becomes highlighted.	As expected.	Pass

2.3.2	The heart button no longer becomes highlighted.	As expected.	Pass
2.4.1	A confirmation dialogue box is displayed, allowing the user to choose “Yes” or “No”.	As expected.	Pass
2.4.2	The dialogue box is closed and the meal is deleted from the database and removed from the list.	As expected.	Pass
2.4.3	The dialogue box is closed without the meal being deleted.	As expected.	Pass
2.5.1	A meal with exactly the given information is added to the database.	As expected.	Pass
2.5.2	All form fields are cleared.	As expected.	Pass
2.6.1	The meal is now listed in the meal browser with the altered name.	As expected.	Pass
2.6.2	Pressing the recipe button next to the meal in the meal browser now directs the user to the altered URL.	As expected.	Pass
2.6.3	The meal is now listed in the meal browser with the new tag present.	As expected.	Pass
2.6.4	The meal is now listed in the meal browser with the new tag present.	As expected.	Pass
2.6.5	The meal is now listed in the meal browser with the removed tag omitted.	As expected.	Pass
2.7.1	The drop-down box is populated with the tags “soup”, “lentil”, “quick” and no more.	As expected.	Pass
2.7.2	The drop-down box is populated with the tags “soup”, “lentil”, “quick” and no more.	As expected.	Pass
2.7.3	The tag appears in the list to the left of the drop-down box.	As expected.	Pass
2.7.4	The tag appears in the list to the left of the drop-down box.	As expected.	Pass
2.7.5	The tag is removed from the list.	As expected.	Pass
2.7.6	No effect.	Not able to remove a tag if the list is empty.	Pass
2.8.1	A single coloured region is shown on the “Nov 2013” page, encompassing the days labelled “3” to “9” inclusive and no further.	As expected.	Pass

2.8.2	Two coloured regions are shown on the “Nov 2013” page. The first encompasses the days labelled “7” to “9”, and the second encompasses those labelled “10” and “11”.	As expected.	Pass
2.8.3	A single coloured region is shown on the “Nov 2013” page, encompassing only the day labelled “7”.	As expected.	Pass
2.8.4	A layout of coloured regions identical to that shown in Figure 2.9 (p. 42) is displayed.	As expected.	Pass
2.8.5	The “Oct 2013” page is now displayed.	As expected.	Pass
2.8.6	The “Dec 2013” page is now displayed.	As expected.	Pass
2.8.7	The “Dec 2012” page is now displayed.	As expected.	Pass
2.8.8	The “Jan 2014” page is now displayed.	As expected.	Pass
2.9.1	A page asking the user to confirm deletion appears, including buttons labelled “Yes” and “No”.	As expected.	Pass
2.9.2	The meal plan is deleted from the database and the user is directed to the meal plan browser.	As expected.	Pass
2.9.3	The user is directed back to the meal plan viewer without the meal plan being deleted.	As expected.	Pass
2.10.1	A new meal plan is created extending from 11 Nov to 16 Nov inclusive, containing no servings.	As expected.	Pass
2.10.2	A new meal plan is created extending from 11 Nov to 16 Nov inclusive, containing 6 servings chosen by the system.	As expected.	Pass
2.11.1	A dialogue box containing a list of meals is presented.	As expected.	Pass
2.11.2	The row is restored to say “Click to change” and the serving is removed from the database.	As expected.	Pass
2.11.3	No effect.	As expected.	Pass
2.11.4	The dialogue box is closed and the serving for the clicked date is updated with the same meal.	As expected.	Pass

2.11.5	The dialogue box is closd without the serving being modified.	As expected.	Pass
2.11.6	The list of servings is reverted to the state it was in when the page was loaded. This change is reflected in the database.	No “Reset” button on form.	Fail
3.1.1	Valid	Valid	Pass
3.1.2	Invalid	Invalid	Pass
3.1.3	Valid	Valid	Pass
3.1.4	Valid	Valid	Pass
3.1.5	Invalid	Invalid; see Figure C.4 (p. 89)	Pass
3.2.1	Valid	Valid	Pass
3.2.2	Invalid	Invalid; see Figure C.5 (p. 89)	Pass
3.2.3	Valid	Valid	Pass
3.2.4	Valid	Valid	Pass
3.2.5	Invalid	Invalid	Pass
3.3.1	Valid	Valid	Pass
3.3.2	Invalid	Invalid; see Figure C.6 (p. 90)	Pass
3.3.3	Invalid	Accepted by client-side validation but rejected by server, leading to “Bad Request” error; see Figure C.7 (p. 90)	Fail
3.3.4	Invalid	Invalid	Pass
3.3.5	Valid	Valid	Pass
3.3.6	Invalid	Valid; date is changed to 01/12/2013	Fail
3.3.7	Valid	Valid	Pass
3.3.8	Invalid	Valid; date is changed to 31/10/2013	Fail
3.3.9	Valid	Valid	Pass
3.3.10	Invalid	Accepted by client-side validation but rejected by server, leading to “Bad Request” error; see Figure C.7 (p. 90)	Fail

3.3.11	Valid	Valid	Pass
3.3.12	Invalid	Accepted by client-side validation but rejected by server, leading to “Bad Request” error; see Figure C.7 (p. 90)	Fail
3.4	*Results are the same as test 3.3*		
3.5.1	Valid	Valid	Pass
3.5.2	Valid	Valid	Pass
3.5.3	Invalid	Invalid; see Figure C.8 (p. 91)	Pass
3.5.4	Valid	Valid	Pass
3.5.5	Valid	Valid	Pass
3.5.6	Invalid	Invalid	Pass
4.1.1	The meal browser displays no meals.	As expected.	Pass
4.1.2	The meal browser displays a single meal with name “Chilli con carne”.	As expected.	Pass
4.1.3	The meal browser displays a single meal with name “Chilli con carne” and recipe URL “http://drive.google.com/”.	As expected.	Pass
4.1.4	The meal browser displays a single meal with name “Chilli con carne” and tags “spicy” and “tomato”.	As expected. See Figure C.9 (p. 91).	Pass
4.1.5	The meal browser displays only the meals “Carrot and lentil soup”, “Chilli con carne” and “Nachos”, in that order.	As expected.	Pass
4.2.1	The user is directed to “http://drive.google.com/”.	As expected.	Pass
4.2.2	The recipe button of the meal appears deactivated.	The recipe button is instead hidden. See Figure C.10 (p. 92).	Pass
4.2.3	The user is directed to “https://drive.google.com/”.	As expected.	Pass
4.3.1	The meal becomes marked as a favourite.	As expected.	Pass
4.3.2	The meal no longer becomes marked as a favourite.	As expected.	Pass
4.4.1	The meal record is deleted from the database.	As expected.	Pass

4.4.2	The meal record and its tag record are both deleted from the database.	As expected.	Pass
4.4.3	The meal record and its tag records are all deleted from the database.	As expected.	Pass
4.5.1	A meal is created with name “Carrot and lentil soup” and recipe URL “http://docs.google.com/”. Tag records “soup”, “lentil” and “quick” are also created and linked to the meal. These attributes are fully displayed in the meal browser.	As expected.	Pass
4.6.1	The ID present in the URL of the destination page is the same as the ID of the meal clicked.	As expected.	Pass
4.6.2	The contents of the “Name of meal” field are “Chilli con carne”.	As expected.	Pass
4.6.3	The contents of the “Link to recipe” field are “http://docs.google.com/”.	As expected.	Pass
4.6.4	The tag list box displays only the tags “lentil”, “quick” and “soup”.	As expected.	Pass
4.7.1	The name of the meal in the database is changed to “Chilli con carne”.	As expected.	Pass
4.7.2	The recipe URL of the meal in the database is changed to “http://drive.google.com/”.	As expected.	Pass
4.7.3	A “tomato” tag record is created for the meal, and the “lentil” tag record is deleted.	As expected.	Pass
4.8.1	The page lists days Fri 15, Sat 16, Sun 17, Mon 18, Tue 19 and Wed 20, with no meals or additional notes.	As expected.	Pass
4.8.2	The page lists days Fri 15, Sat 16, Sun 17, Mon 18, Tue 19 and Wed 20, with no meals. The additional notes read “These are some example notes”.	As expected.	Pass
4.8.3	The serving list shows only Fri 15.	As expected.	Pass
4.8.4	The page lists days Fri 15, Sat 16, Sun 17, Mon 18, Tue 19 and Wed 20, with the selected meal displayed next to Mon 18.	As expected. See Figure C.11 (p. 92).	Pass
4.9.1	The page lists days Fri 15, Sat 16, Sun 17, Mon 18, Tue 19 and Wed 20, with no meals or additional notes.	As expected.	Pass

4.9.2	The page lists days Fri 15, Sat 16, Sun 17, Mon 18, Tue 19 and Wed 20, with no meals. The contents of the "Notes" field read "These are some example notes".	As expected.	Pass
4.9.3	The serving list shows only Fri 15.	As expected.	Pass
4.9.4	The page lists days Fri 15, Sat 16, Sun 17, Mon 18, Tue 19 and Wed 20, with the selected meal displayed next to Mon 18.	As expected.	Pass
4.10.1	The meal plan record is deleted from the database.	As expected.	Pass
4.10.2	The meal plan record and the serving record is deleted from the database.	As expected.	Pass
4.10.3	The meal plan record and the serving records are all deleted from the database.	As expected.	Pass
4.11.1	A meal plan record spanning 15–20 Nov 2013 is created.	As expected.	Pass
4.11.2	A meal plan record spanning 15–20 Nov 2013 and with notes "These are some example notes" is created.	As expected.	Pass
4.11.3	A meal plan record spanning only 15 Nov 2013 is created.	As expected.	Pass
4.12.1	A serving record for 18 Nov 2013 and the selected meal is added to the meal plan.	As expected.	Pass
4.12.2	The previously added serving record is deleted and a new one for the new meal is added.	As expected.	Pass
4.13.1	The serving record is deleted from the database.	As expected.	Pass
4.14.1	The meal plan is reverted to the state it was in before the change.	No "Reset" button present on form.	Fail
4.14.2	The meal plan is reverted to the state it was in before the change.	No "Reset" button present on form.	Fail

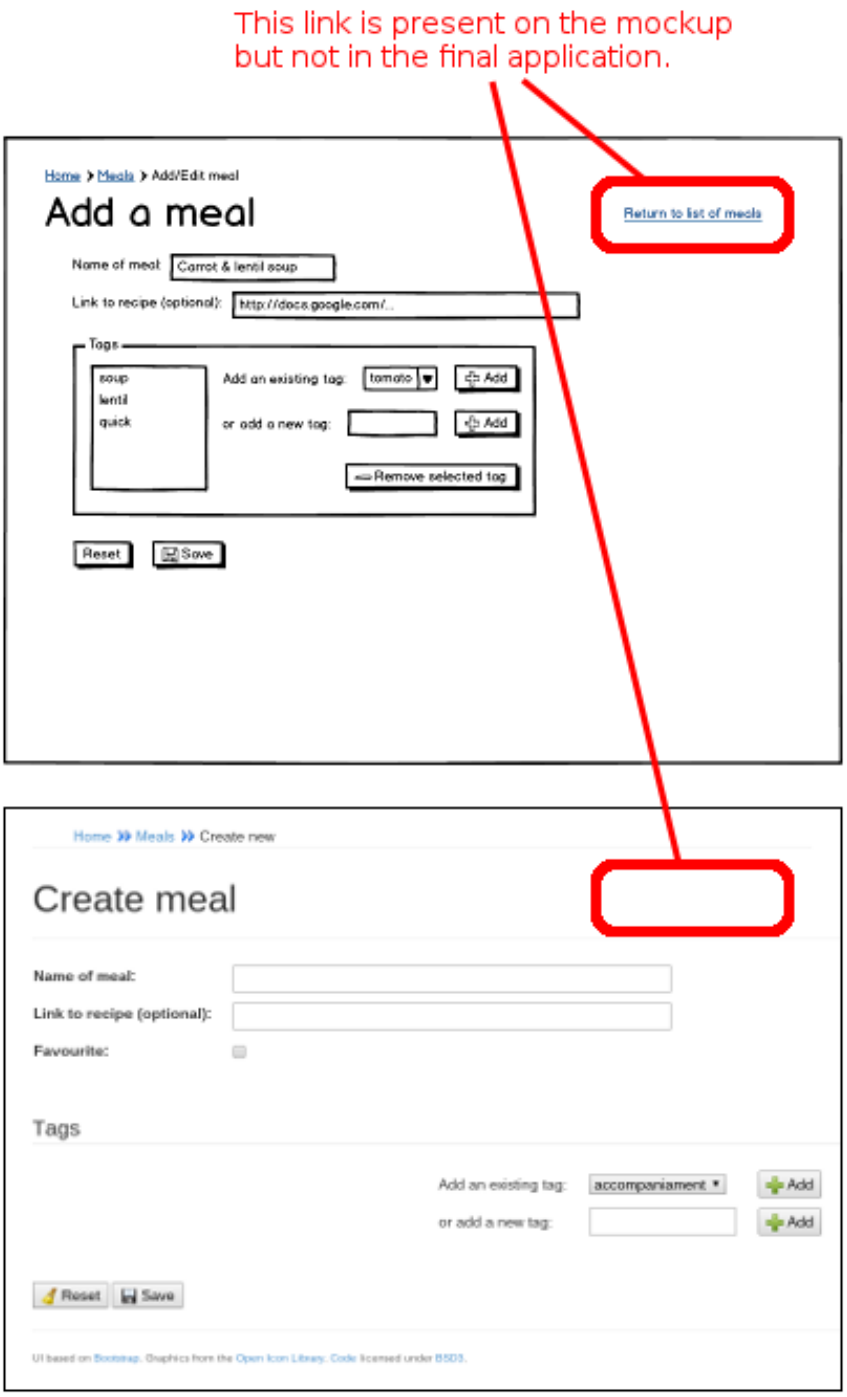

Figure C.1 – Tests 1.3.3 and 1.4.3: Missing "Return to list of meals" link on meal creation/editing page

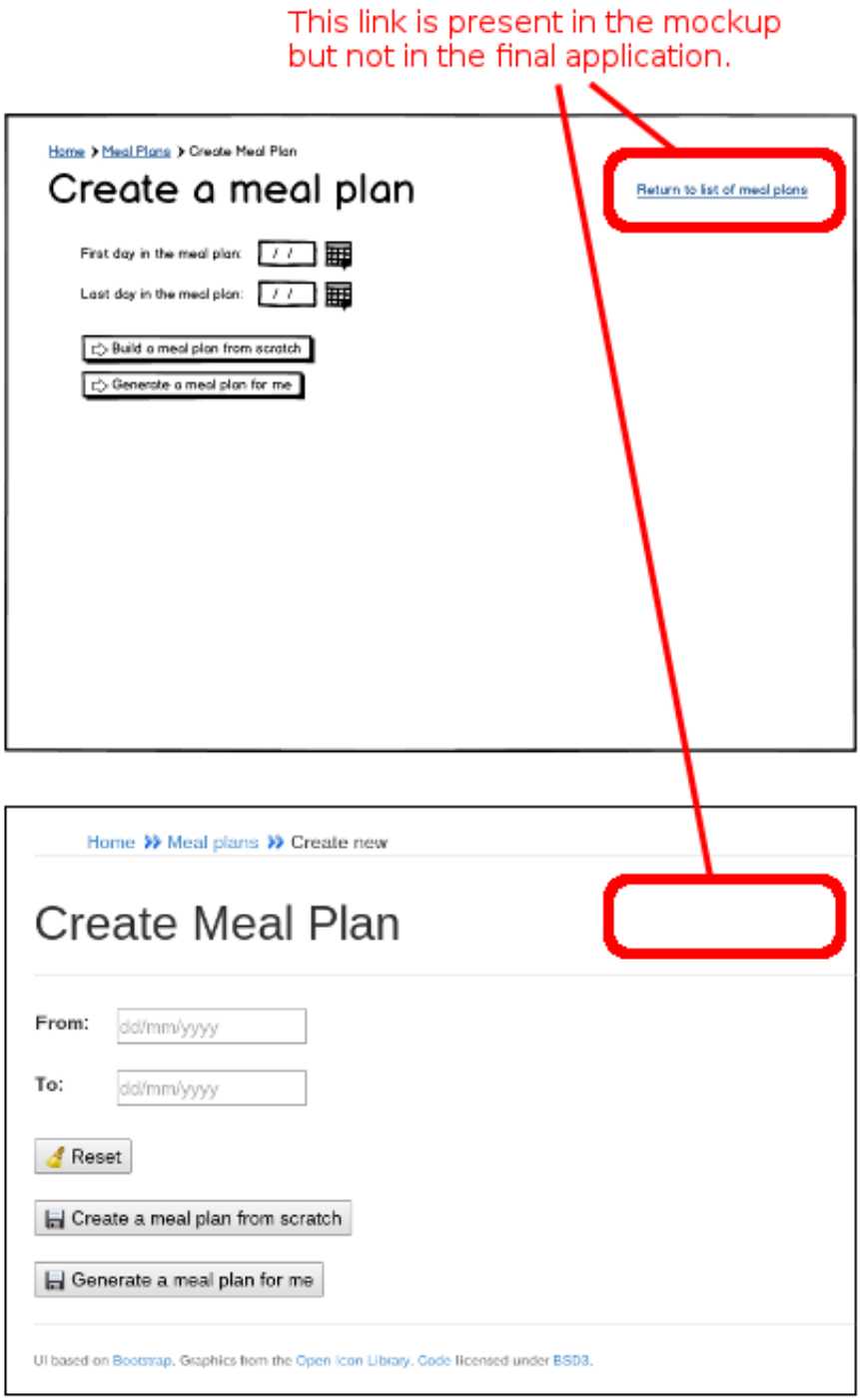

Figure C.2 – Test 1.7.3: Missing "Return to list of meals" link on meal plan creation page

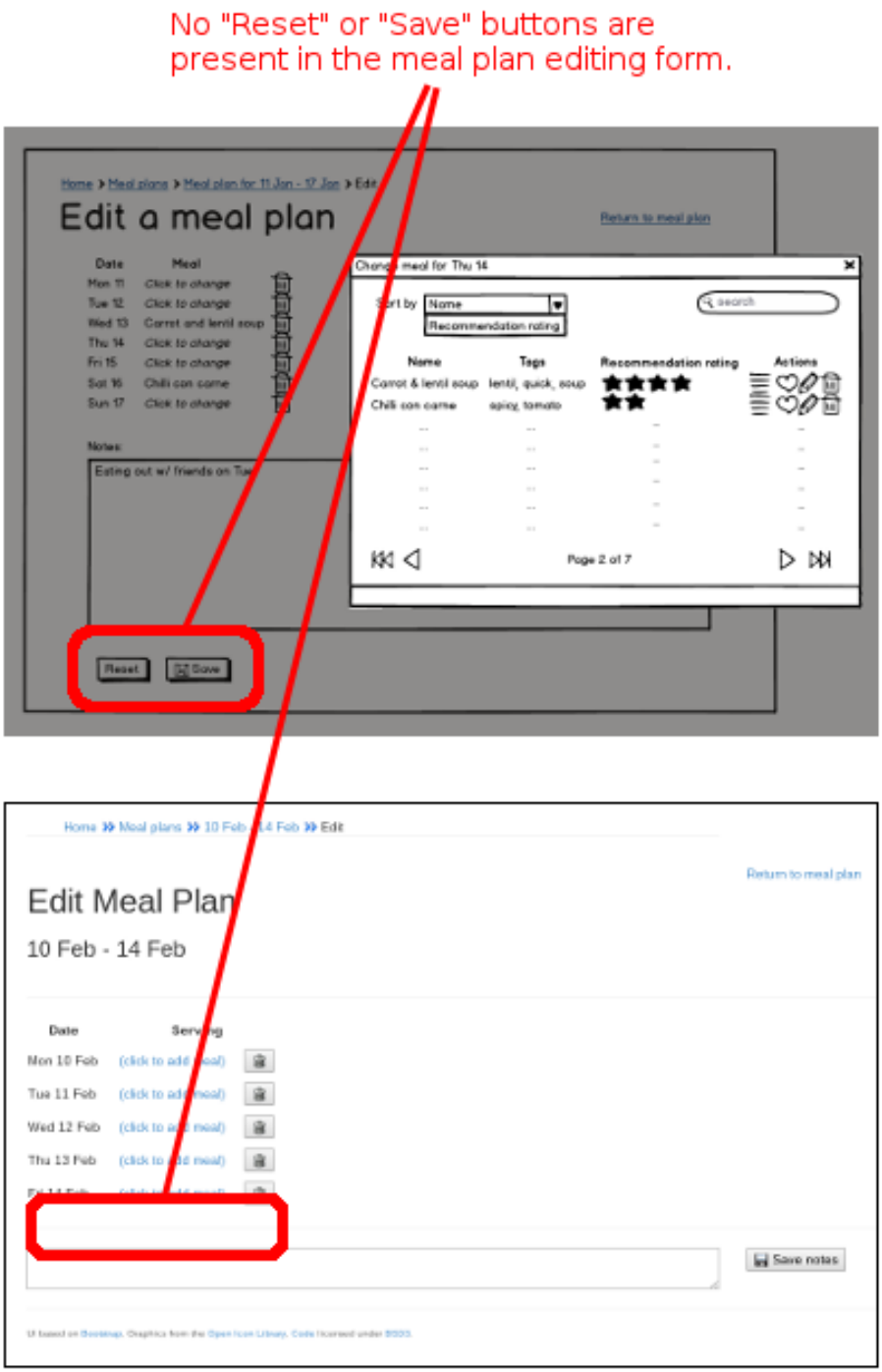

Figure C.3 – Test 1.8.5: Missing “Save” button on meal plan editor page

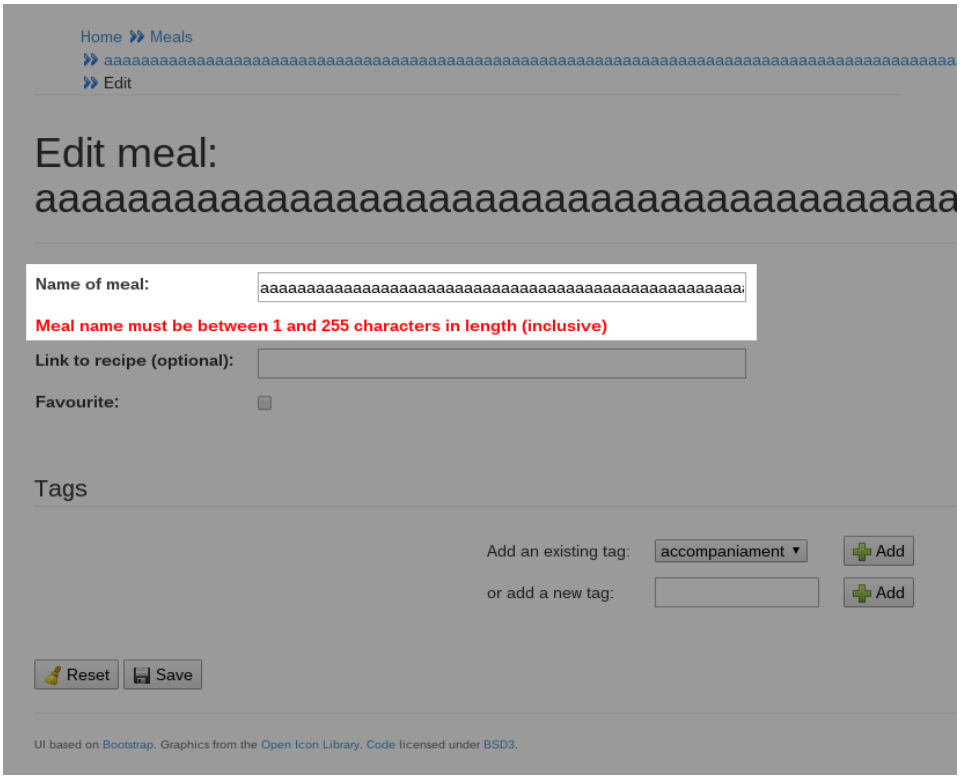

Figure C.4 – Test 3.1.5: Correct validation of 256 character meal name

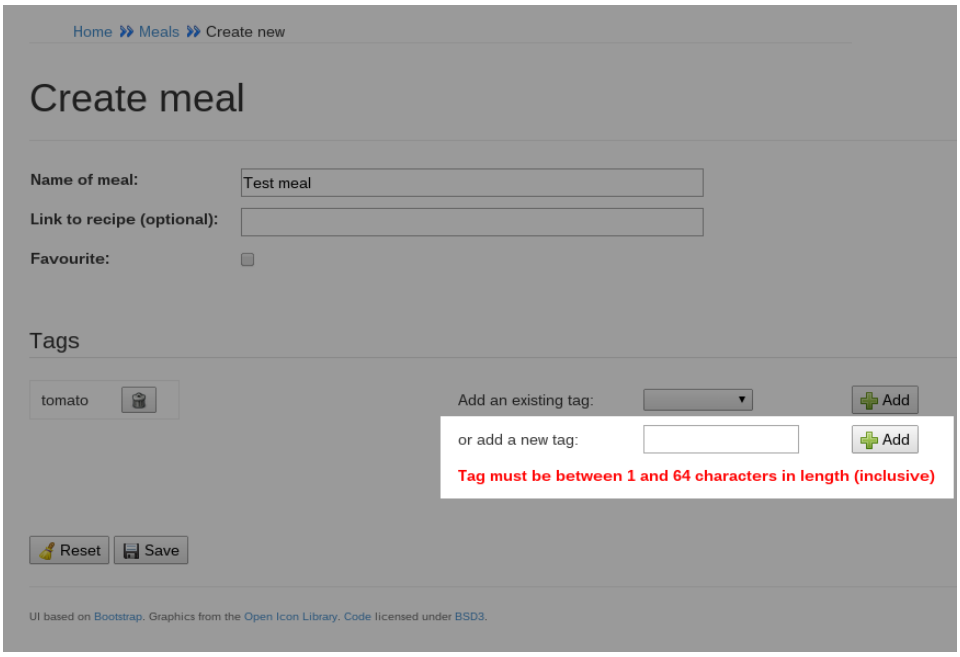

Figure C.5 – Test 3.2.2: Correct validation of empty tag name

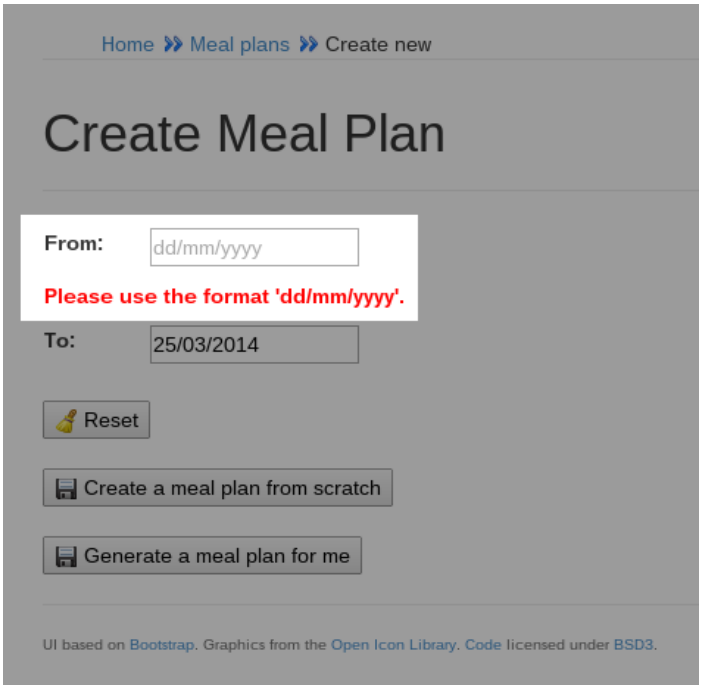

Figure C.6 – Test 3.3.2: Correct validation of empty meal plan date

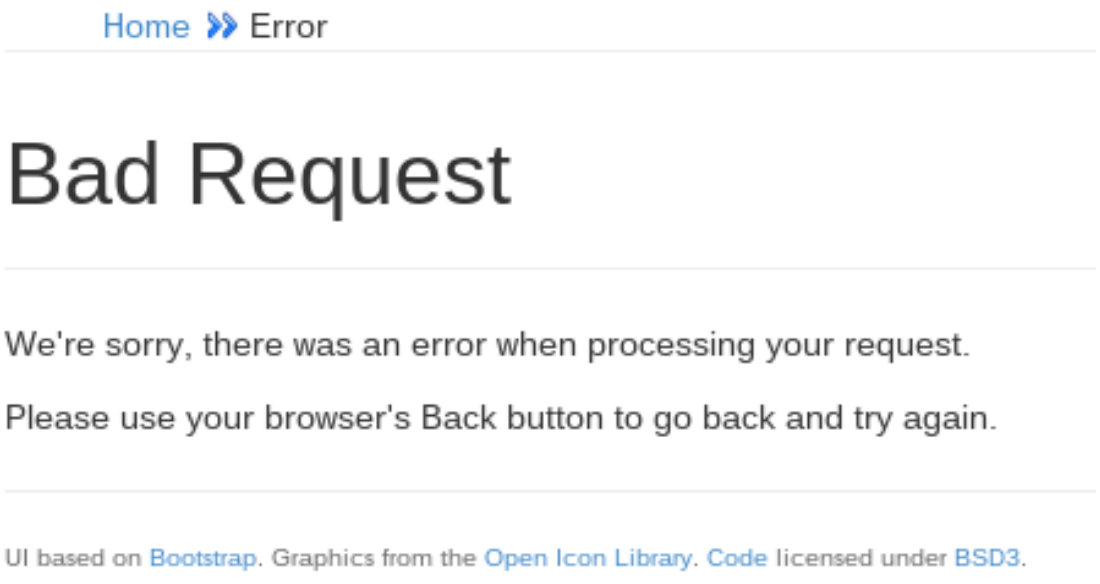

Figure C.7 – Test 3.3.3: "Bad Request" error caused by the invalid date string "2013/11/15"

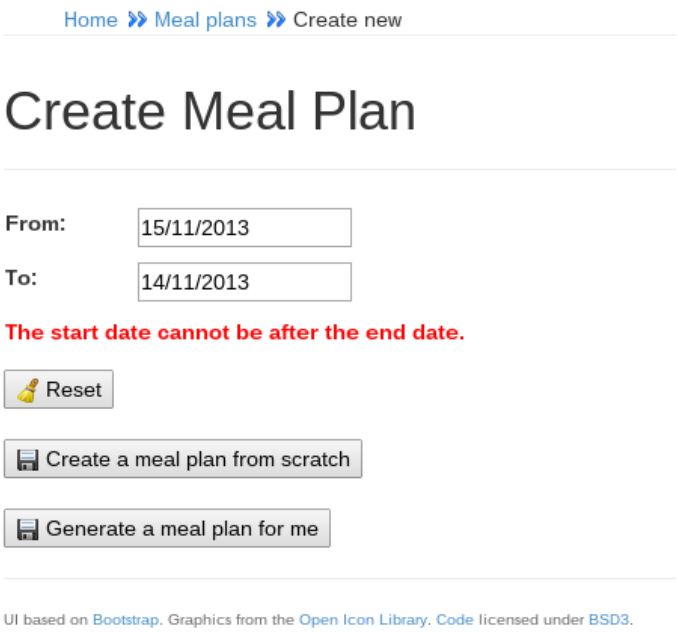

Figure C.8 – Test 3.5.3: Correct validation of meal plan dates

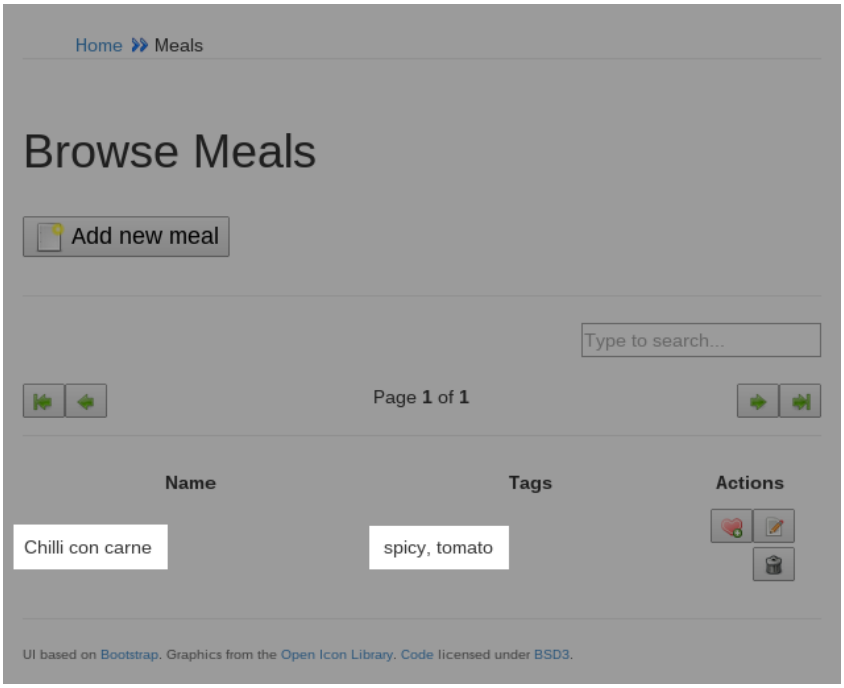

Figure C.9 – Test 4.1.4: Meal correctly shown in list

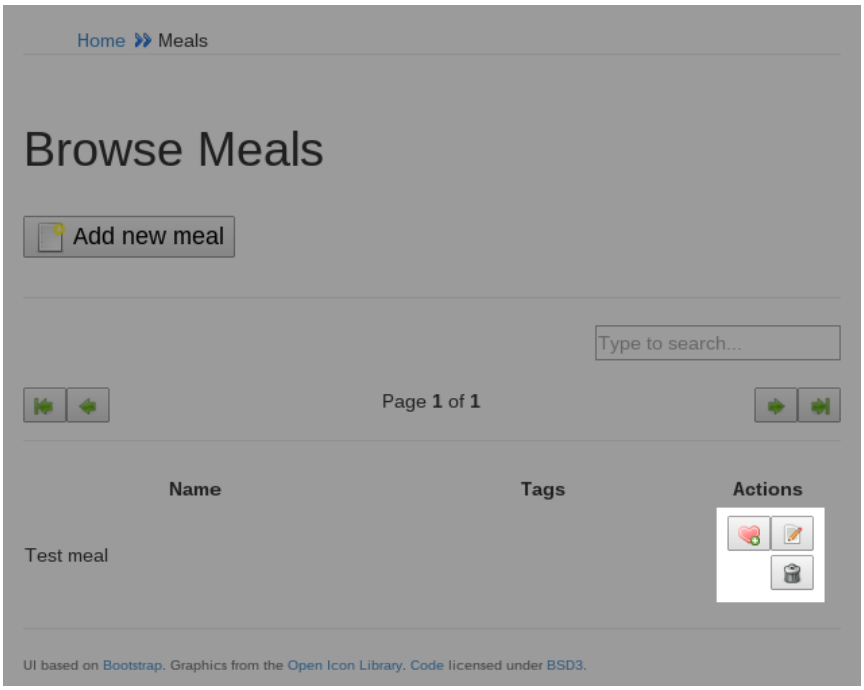

Figure C.10 – Test 4.2.2: Recipe button hidden for meals without a recipe URL

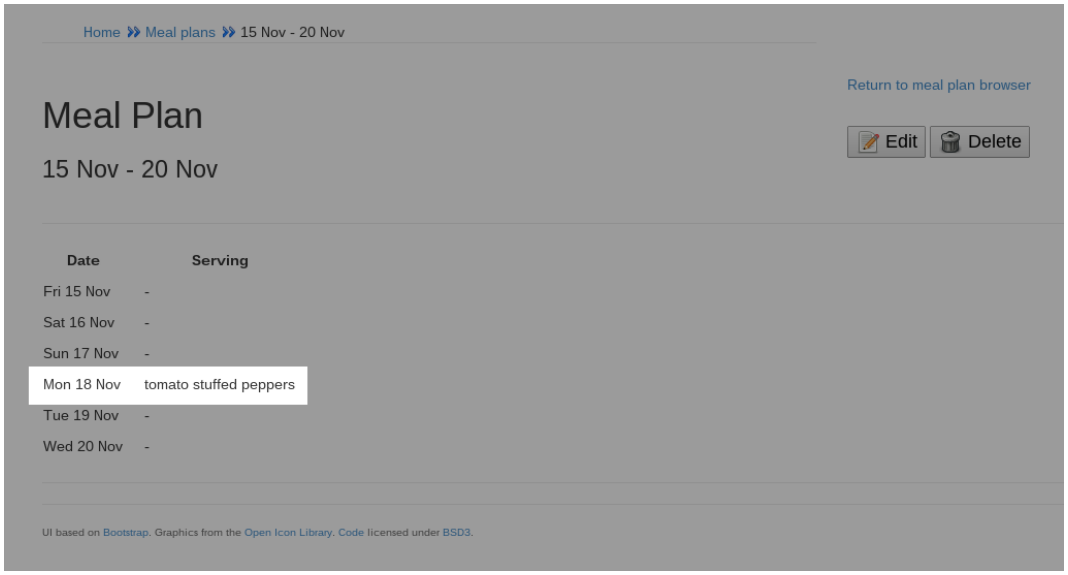

Figure C.11 – Test 4.8.4: Meal plan viewer, showing serving on 18 Nov 2014

D. User manual

Meal Planner User Manual

Kier Davis

27th November 2015

Contents

List of Figures

1 Introduction

The Meal Planner is designed to be a simple yet powerful application to assist in planning meals for the week. It allows you to manage a list of meals, from which suggestions are drawn that you can add to meal plans.

This document aims to help you to get set up using the Meal Planner and guide you through the most common tasks you may encounter when using it.

2 Setup

The Meal Planner server software depends on an external MySQL-compatible[1] database to function; this can either be installed on your local computer along with the Meal Planner software, or on a remote server.

2.1 Setup on Microsoft Windows

2.1.1 Installing the software

The software can be installed on Microsoft Windows systems by following these steps:

1. Download and install the MySQL Community Edition software. Comprehensive instructions on how to do this are given at `http://dev.mysql.com/doc/refman/5.6/en/installing.html`.

2. Download the Meal Planner software installer from the download page.[2] Figure 2.1 (p. 5) shows the download links that you will need - choose *one* depending on whether your operating system is a 32-bit or 64-bit one. If you're unsure which yours is, the 32-bit installer will work on both.

   - If you are asked whether you want to save or run the file, click "run" (the dialog box may look like the one shown in Figure 2.2 (p. 5)).
   - If you are asked to confirm that you want to run the program, click "yes" / "run" (the dialog box may look like the one shown in Figure 2.3 (p. 5)).

3. Start the installer, if it did not start automatically after the download completed. You should be presented with the window shown in Figure 2.4 (p. 5).

4. Press "Next". The installer will ask you to specify a folder to install the software into. The default is usually suitable, however any folder will work. This screen is shown in Figure 2.5 (p. 5).

5. Press "Next" again. The installer will ask you to configure the database. This window is shown in Figure 2.6 (p. 6). The fields are explained as follows:

   Server address The hostname or IP address of the server the database is running on. If the database is running on the local machine, enter "localhost".

[1] See also: `http://www.mysql.com/`

[2] Meal Planner download page: `https://github.com/kierdavis/mealplanner/tree/releases`

Server port The port number that the database is listening on. Unless you have changed this in your database's settings, it is most likely 3306.

Note: make sure that if the database is running on a remote machine, this port is marked open in the machine's firewall configuration.

Username The username of the user to connect to the database as. You may have created a database user as part of the database installation process; if not, there are many guides online on how to do this.

Password The password of the user to connect to the database as.

Note that this password is not masked in this text field, and is stored unencrypted on your computer, so it is not recommended to use one that you use for other applications. A randomly generated key is considered most secure.

Database name The name of the database that the Meal Planner will use to store its data in (since one MySQL server allows many databases to be stored, each identified by a name).

6. Press "Next" again. The installer will now ask you to select the name of Start Menu folder to place shortcuts in. Generally, the default is fine. This screen is shown in Figure 2.7 (p. 6).

7. Press "Next" again. The installer will begin the installation process. This may take a couple of minutes to complete.

   Once it finishes, press "Next" once more. The installer is now finished and can be closed. This final screen is shown in Figure 2.8 (p. 6).

2.1.2 Running the server

The server can be started from the Start Menu. A folder on the Start Menu should have been created during the installation procedure; it will be called "Meal Planner" unless you changed the name from the default. The folder should contain an item named "Start Server", which will open the Meal Planner server software when it is clicked. The location of this item is shown in Figure 2.9 (p. 6).

It is possible that a dialogue box like the one in Figure 2.10 (p. 6) may appear. This is a result of Microsoft Windows Firewall asking if you want to allow the Meal Planner software to listen for incoming connections on your computer. If this happens:

1. Make sure that the checkbox labelled "Private networks" is checked.

2. Make sure that the checkbox labelled "Public networks" is *not* checked.

3. Click "Allow access". This operation requires administrator privilege (as indicated by the yellow/blue shield icon on the button), so if your Windows user account is not an adminstrator one you may be asked to enter an administrator password or log in to an administator account.

The application can then be accessed by navigating to `http://localhost/` in a browser (or, from a remote computer, `http://<hostname>/` where `<hostname>` is the host name or IP address of the computer running the server).

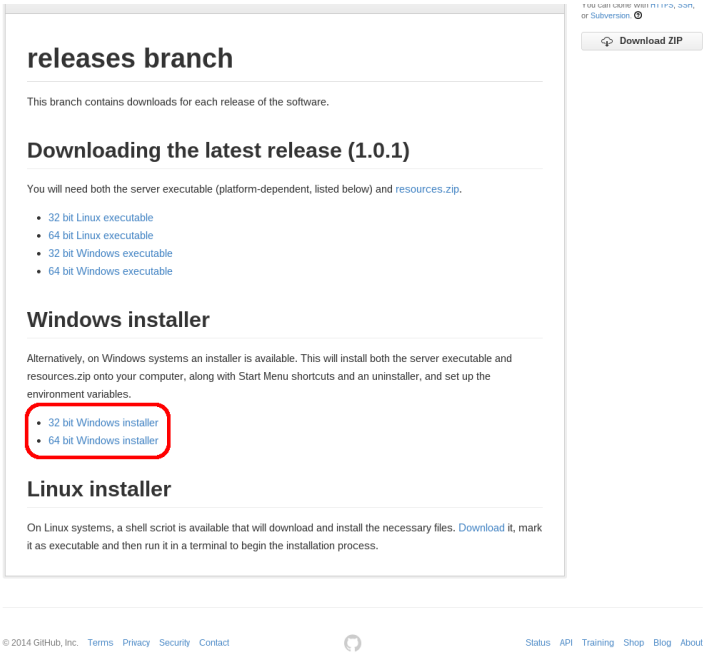

Figure 2.1 – Releases page, with Windows installer downloads highlighted

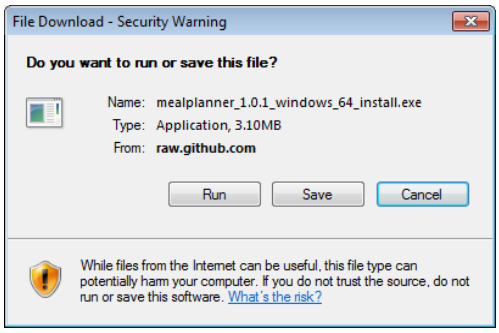

Figure 2.2 – "Save or run" dialog

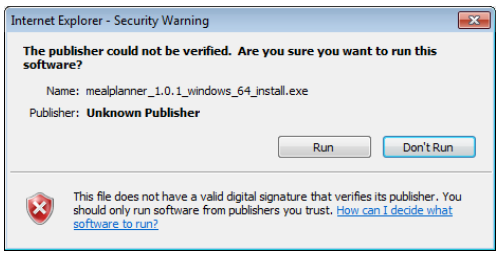

Figure 2.3 – "Confirm run" dialog

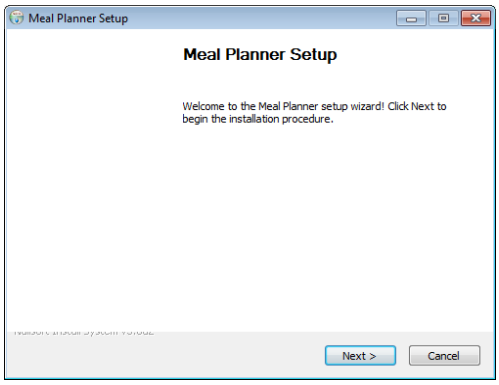

Figure 2.4 – Welcome page of installer

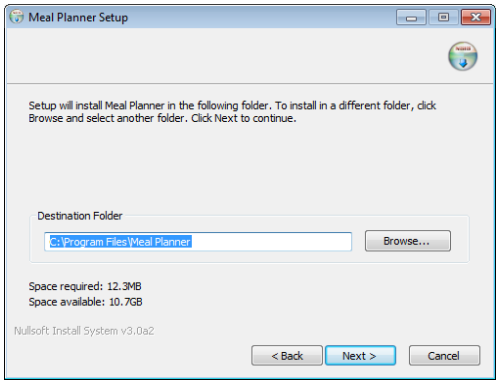

Figure 2.5 – Destination folder selection page of installer

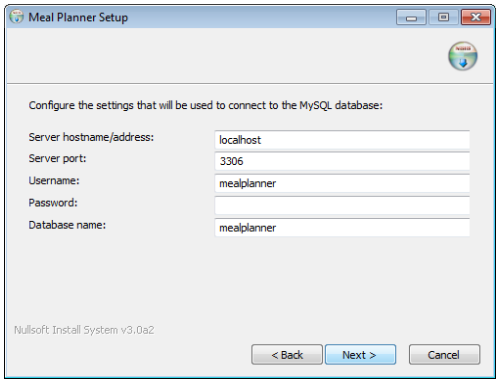

Figure 2.6 – Database configuration page of installer

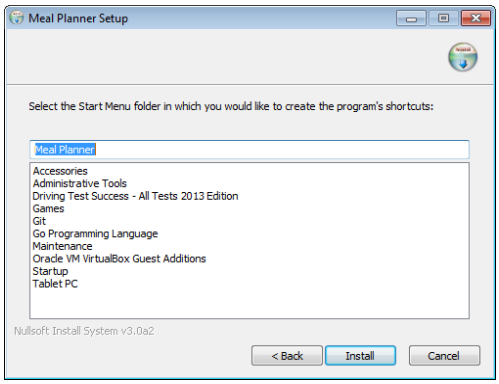

Figure 2.7 – Start Menu folder selection page of installer

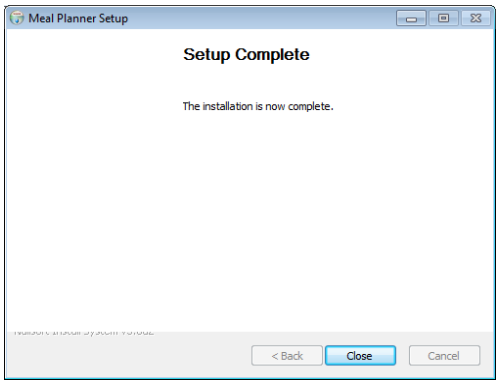

Figure 2.8 – Completion page of installer

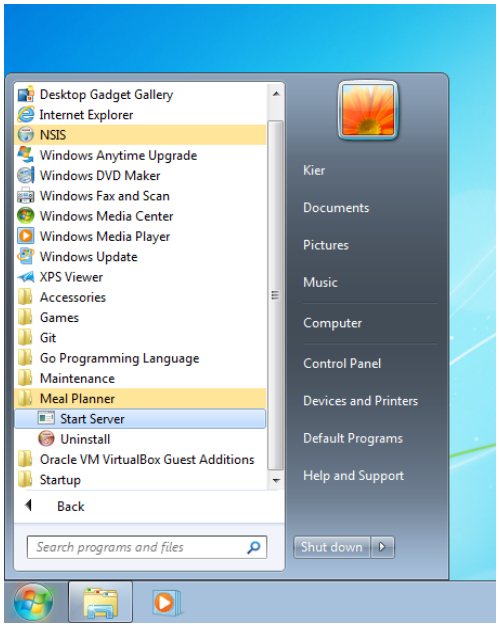

Figure 2.9 – Location of "Start Server" item on Start Menu

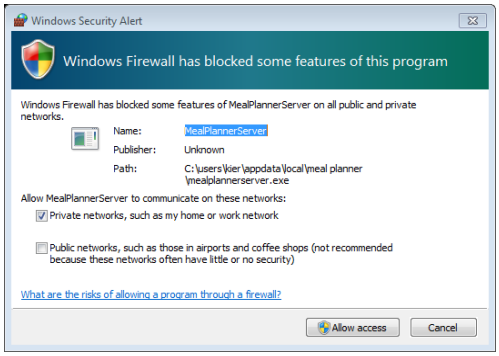

Figure 2.10 – Microsoft Windows Firewall dialogue box

2.2 Setup on Linux

2.2.1 Installing the software

The software can be installed on Linux systems by following these steps:

1. Download and install a MySQL-compatible server. The official MySQL software and/or the enhanced MariaDB software can be found in most package manager repositories.
2. Download the Meal Planner software installer script from the download page.[3] Figure 2.11 (p. 8) shows the download link that you will need.
3. Once the download is complete, mark the installer script as executable. This can be done from a terminal with `chmod +x mealplanner_1.0.1_linux_install.sh`.
4. Run the installer script. The Meal Planner software will be downloaded and installed. You may be prompted to enter your system password in order to install software to your system.
5. You will then be asked to enter the database information. For an explanation of the fields see step 5 of the Windows installation procedure in Section 2.1.1 (p. 3). If you leave a field blank the default in the square brackets will be used.
6. The installer will now list two shell script lines that must be added to a startup script. This can be done by editing (or creating) the file `/home/<username>/.bashrc` and adding these two lines to the bottom.
7. The installation is now complete.

2.2.2 Running the server

The server can be started by entering `mealplanner-server` into a terminal. By default the server listens on port 80, which on some systems will require you to run it as the superuser. To do this, enter `sudo mealplanner-server` instead and enter your login password when asked.

The application can then be accessed by navigating to `http://localhost/` in a browser (or, from a remote computer, `http://<hostname>/` where `<hostname>` is the host name or IP address of the computer running the server).

[3] Meal Planner download page: `https://github.com/kierdavis/mealplanner/tree/releases`

3 Getting started

3.1 Adding meals

The first thing you may want to do is to add meals to the database. This can be done by following these steps:

1. Navigate to the homepage by clicking the "Home" link at the top left-hand corner of any page.
2. Press "Add a meal". The form shown in Figure 3.1 (p. 8) should display.
3. Enter a name for the meal, such as "Mushroom soup".
4. Optionally, enter the URL/address of a webpage for the recipe for the meal. For example, you can copy and paste the contents of the address bar when viewing a Google Docs document into this field.
5. Meals can be marked as whether they are a favourite or not. Favourite meals are more likely to be suggested by the application when creating or editing a meal plan. If you consider this meal a favourite, tick the checkbox labelled "Favourite".
6. Meals can also be tagged to put them into groups. Meals are less likely to be suggested if their tags are similar to the tags of meals also being served around the same time. Examples of tags could be:
   - types of dish such as "soup" or "pie"
   - major components of the dish such as "lentil" or "tomato"
   - what the dish is often served with such as "rice" or "salad"

   The interface allows you to add a tag from a list of ones you have previously added to meals, or to add a completely new one. Tags can be deleted from the list by pressing the button with the icon of a trash can next to the tag.
7. When you are done, press Save. The meal will be added and you will be able to see it in the list. Another meal can be added by pressing the "Add meal" button in the top right-hand corner of this screen.

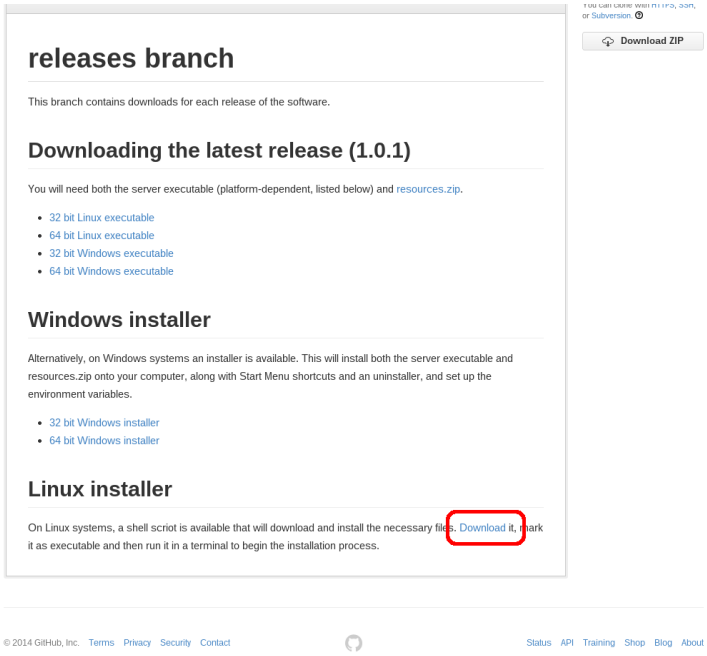

Figure 2.11 – Releases page, with Linux download highlighted

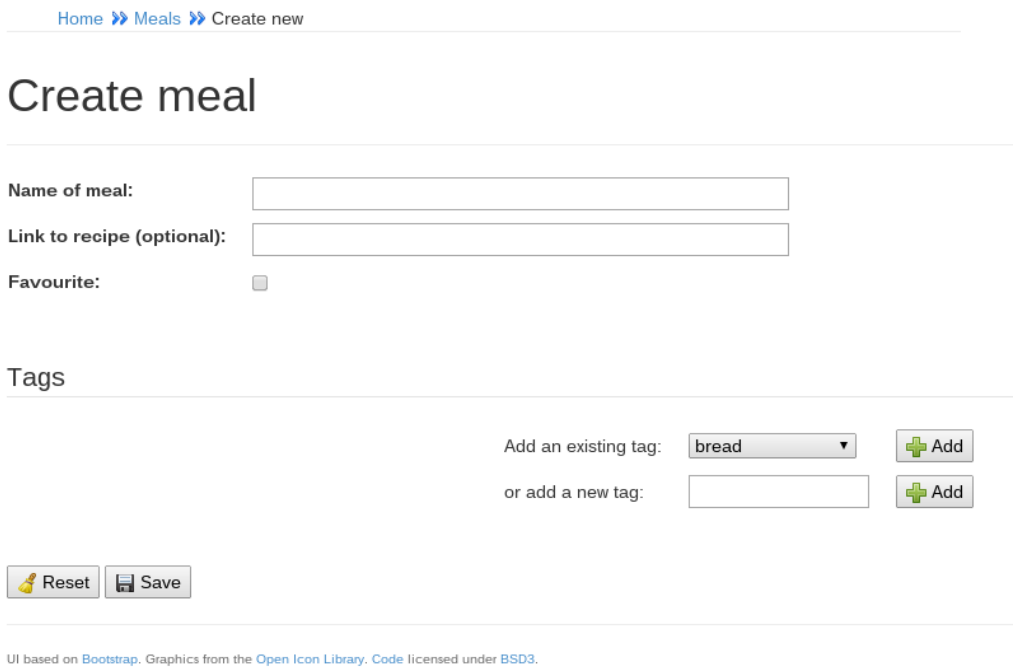

Figure 3.1 – Meal creation page

3.2 Searching for meals

Meals can be searched for based on their name, recipe URL and tags. You can search the list of meals by following these steps:

1. Navigate to the homepage by clicking the "Home" link at the top left-hand corner of any page.
2. Press "Browse meals". The page shown in Figure 3.2 (p. 10) should display.
3. Select the search box (near the top-right) and enter your search query. The application will automatically search the database when you stop typing; there is no need to press the Enter key.
4. Deleting the contents of the search box will cancel the search and return the table to displaying all meals.
5. Each meal in the list has three or four buttons to its right. They are:
   - The first button, with the icon of a book, will open the recipe URL if you added one when you created or edited the meal. If you didn't add a recipe URL, this button will not be shown.
   - The second button, with the icon of a heart, will mark or unmark the meal as a favourite. If the meal is not currently a favourite, the icon will have a small green plus sign in the corner and clicking it will mark it as a favourite. Likewise, if the meal is a favourite the icon will have a small red minus sign in the corner and it will unmark the meal as a favourite when clicked.
   - The third button, with the icon of a pencil on paper, will open a form that can be used to edit the information about the meal. It is almost exactly the same as the form you used when creating a meal; see Section 3.1 (p. 7) for instructions on using this form.
   - The fourth and final button, with the icon of a trash can, will delete the meal from the database when clicked.

3.3 Creating meal plans

You can create a meal plan by following these steps:

1. Navigate to the homepage by clicking the "Home" link at the top left-hand corner of any page.
2. Press "Create a meal plan". The page shown in Figure 3.3 (p. 10) should display.
3. Meal plans are created by specifying a range of days that are next to each other. Click in the field labelled "Start date" to bring up a date selection pop-up that can be used to enter the first day in this range of days. Do the same for the "End date" field.

   For example, Figure 3.4 (p. 11) shows the dates you would enter for a meal plan that spans from 17 March to 21 March 2014.
4. You can now either press "Create a meal plan from scratch" or "Generate a meal plan for me".
   - Pressing "Create a meal plan from scratch" will create an empty meal plan and allow you to add your own servings to it.
   - Pressing "Generate a meal plan for me" will create a meal plan and automatically add servings to it, using the highest-rated suggestion for each day.
5. After creating the meal plan you will be redirected to a page like the one shown in Figure 3.5 (p. 12) that will allow you to edit the meal plan. It contains a table listing each day in your meal plan and the meal assigned to that day.
6. A meal can be assigned to a day by clicking the "(click to change)" text if there is no meal currently assigned, or the name of the meal if there is a meal currently assigned. This should open a popup box like the one in Figure 3.6 (p. 12).

   The meals shown in this box are sorted by their *score*. This is a numerical value indicated how strongly the application suggests that you should serve this meal on this day. It is calculated based on a number of factors, including:
   - how long it has been since the meal was last served / how long it will be until it is next served
   - how similar the meal is to other meals served around this time (based on tags)
   - whether or not the meal is marked as a favourite

   Because of this, suggestions will be slightly different for each day, and adding/removing servings of a meal from a meal plan can affect the suggestions for other days.

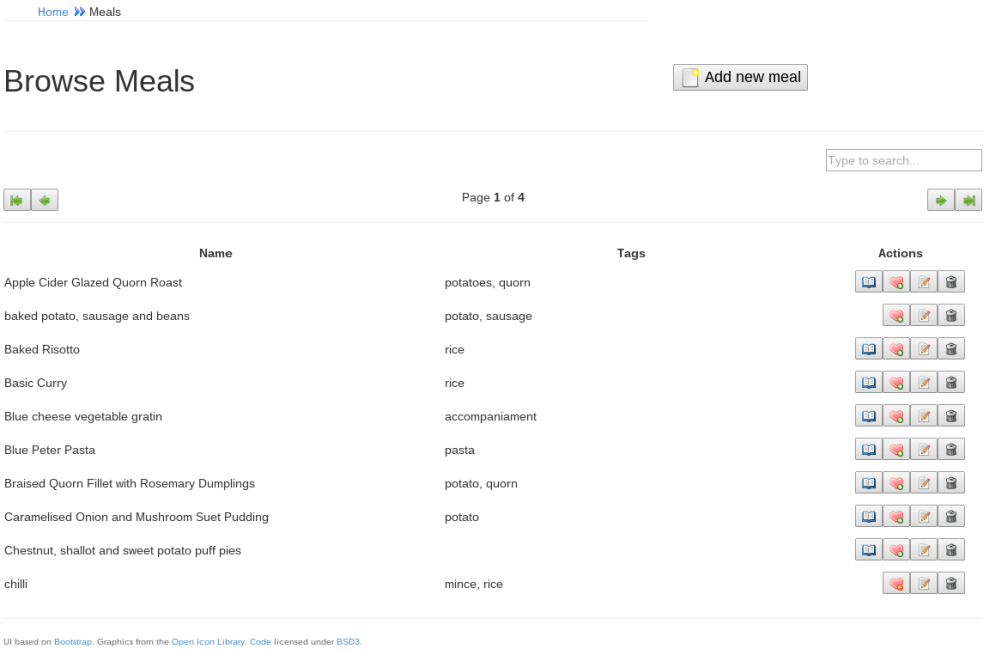

Figure 3.2 – Meal list page

7. To select a meal from this popup box, click the name of the meal in the list. The box should close and the serving of the meal is added to the meal plan, replacing any existing serving for that day.

8. To remove a serving for a day (returning it to the "click to change" state), press the button with the icon of a trash can next to the serving.

9. There is a text box at the bottom that can be used to keep arbitrary notes about the meal. Things you could consider putting here include:

   - who is away from home on certain days
   - any guests that may be coming round
   - any special events that may affect the planning of your meals

   Important: these notes will not save when leaving the page; you need to click the "Save notes" button to the right of or below the text box.

10. When you are done, click "Return to meal plan" in the top right-hand corner of the page to return to viewing the meal plan.

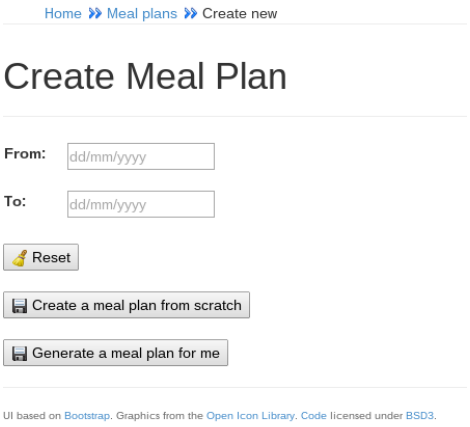

Figure 3.3 – Meal plan creation page

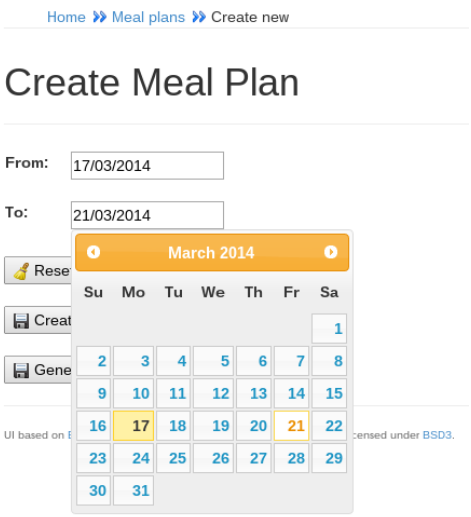

Figure 3.4 – Meal plan creation page, with dates filled in for 17 March to 21 March, 2014

4 Troubleshooting

There are circumstances in which the application may not behave as you would expect. The most common problems and guidance on how to solve them are given in this section.

4.1 Server window disappears immediately after opening

This is usually caused by the server encountering an unrecoverable error during startup, causing the window to be closed before you have time to read the error message it produces. The error message produced is not diagnosed in this section; see Section 4.2. This section will also only discuss this event's occurrence in Microsoft Windows, as the installation instructions above for Linux only described running the server from the command line anyway.

To see the error message that the server produces, we will run the server from the command line instead of as an application with its own window. This can be done as follows:

1. On newer Microsoft Windows systems (those since Windows Vista), a command prompt can be opened by opening the Start Menu, entering "cmd" into the search box and pressing Enter. This is illustrated in Figure 4.1 (p. 12).

   On older Microsoft Windows (those up to Windows XP), a command prompt can be opened by opening the Start Menu, pressing Run, entering "cmd" into the text field and pressing Enter. The location of the "Run" button is shown in Figure 4.2 (p. 13).

2. Once the command prompt has opened, type the following command (including the quotes) and press Enter:

   ```
   "AppData\Local
   Meal Planner
   MealPlannerServer.exe"
   ```

   If you specified a different installation location during setup, replace `AppData Local Meal Planner` with the path to your installation location.

3. The server should then run, allowing any error messages printed to the screen to be seen. Read on to the next section to diagnose these.

4.2 Command-line server errors

4.2.1 "Please specify a non-empty -dbsource flag"

This error, also depicted in Figure 4.3 (p. 13), is caused by the server program not having been informed how to connect to the database. The database connection settings can be specified to the program in two ways: either setting the environment variable named `MPDBSOURCE` (this is set globally by both the Windows and Linux installers) or by specifying it on the command line with the `-dbsource` flag (this way overrides the first).

To fix this, you can specify the database source when running the program. The database source is a string with the format:

`<user>:<pass>@tcp(<host>:<port>)/<name>`

where `<user>` and `<pass>` are the username and password of the user to connect with, `<host>` is the hostname of the server, `<port>` is the port to connect to the server on and `<name>` is the name of the database. For more information on the database settings, see step 5 of the Windows setup guide in Section 2.1.1 (p. 3).

The database source can be passed to the server by running the command `set MPDBSOURCE="<dbsource>"` (Windows) or `export MPDBSOURCE="<dbsource>"` (Linux), before starting the server in the same command prompt window.

4.2.2 "Resource directory not set"

This error, shown in Figure 4.4 (p. 13), is caused by the server program not having been informed where to find the resource files. Like the database connection settings, it can be specified either in the environment variable named `MPRESDIR` (set globally by the installers) or by specifying it on the command line with the `-resourcedir` flag (this way overrides the first).

To fix this, you can specify the resource directory when running the program. This is the path to the folder named `resources` that is extracted from the `resources.zip` download. It can be passed to the

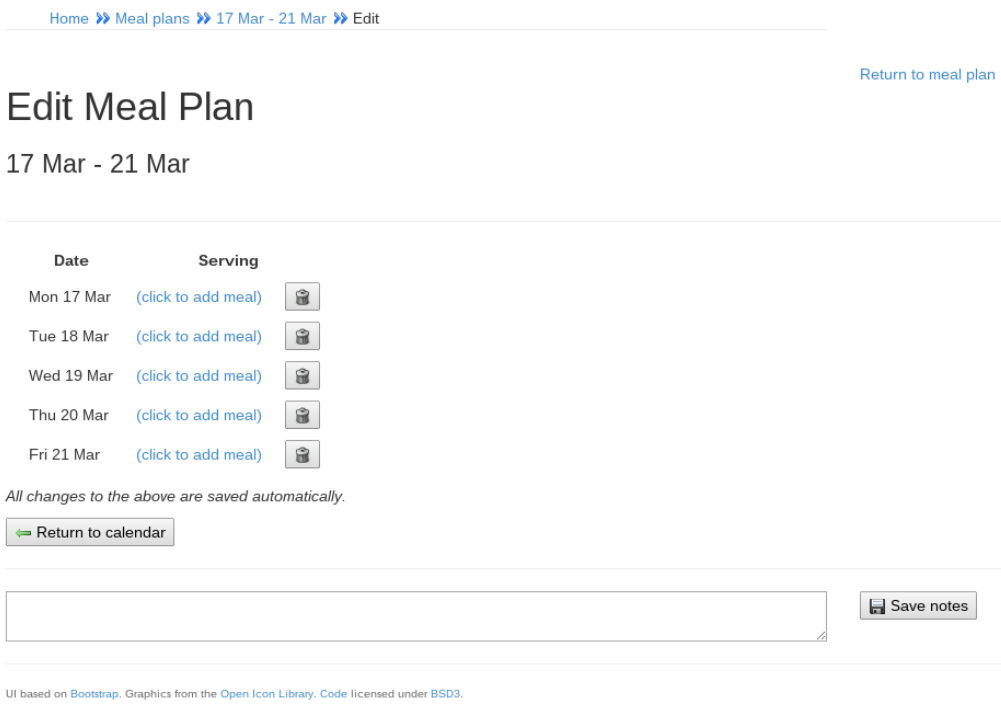

Figure 3.5 – Meal plan editor

Figure 3.6 – Pop-up meal selection box in meal plan editor

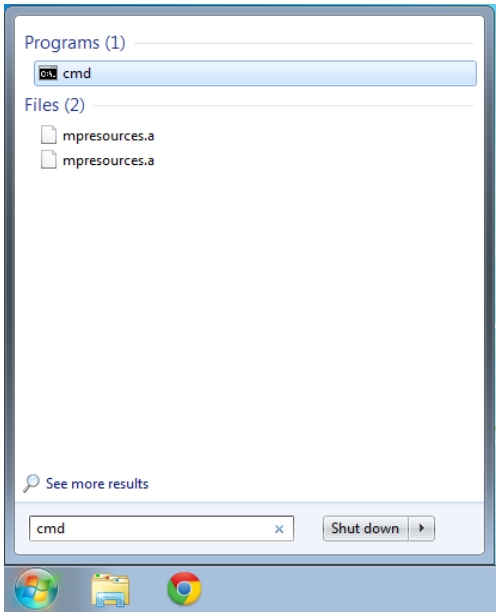

Figure 4.1 – Opening the command prompt on newer Windows systems

Figure 4.2 – Location of the Run button on older Windows systems

```
C:\Users\Kier\AppData\Local\Meal Planner>MealplannerServer.exe
Please specify a non-empty -dbsource flag or set the MPDBSOURCE environment vari
able.
```

Figure 4.3 – "Please specify a non-empty -dbsource flag or set the MPDBSOURCE environment variable" error

```
C:\Users\Kier\AppData\Local\Meal Planner>MealPlannerServer.exe
panic: Resource directory not set and no suitable directory found in the GOPATH

goroutine 1 [running]:
runtime.panic(0x62f580, 0x11bd0710)
        C:/Users/ADMINI~1/AppData/Local/Temp/2/makerelease529514319/go/src/pkg/r
untime/panic.c:266 +0xa6
github.com/kierdavis/mealplanner/mpresources.getSourceDir(0x0, 0x0)
        c:/Users/Kier/mpbuild/v1.0.1/src/github.com/kierdavis/mealplanner/mpreso
urces/mpresources.go:21 +0x23
github.com/kierdavis/mealplanner/mpresources.GetResourceDir(0x0, 0x0)
        c:/Users/Kier/mpbuild/v1.0.1/src/github.com/kierdavis/mealplanner/mpreso
urces/mpresources.go:31 +0x42
github.com/kierdavis/mealplanner/mpresources.GetTemplates(0x6d4550)
        c:/Users/Kier/mpbuild/v1.0.1/src/github.com/kierdavis/mealplanner/mpreso
urces/mpresources.go:53 +0x33
main.main()
        c:/Users/Kier/mpbuild/v1.0.1/src/github.com/kierdavis/mealplanner/mealpl
anner.go:46 +0x140
```

Figure 4.4 – "Resource directory not set and no suitable directory found in the GOPATH" error

server by running the command
`set MPRESDIR="<path>"` (Windows) or
`export MPRESDIR="<path>"` (Linux), before starting the server in the same command prompt window.

4.2.3 "Database error during startup"

This error, an example of which is shown in Figure 4.5, is caused by a failure to connect to the database server.

```
C:\Users\Kier\AppData\Local\Meal Planner>MealPlannerServer.exe
2014/03/17 10:14:18 Database error during startup: dial tcp 127.0.0.1:3306: Conn
ectEx tcp: No connection could be made because the target machine actively refus
ed it.
```

Figure 4.5 – An example of a "Database error during startup" error

A number of possible reasons for this error are listed below:

- You have typed in the database connections settings incorrectly. To fix this, either reinstall the application (if you used the installer) or change the contents of the environment variable, if you set the variable by hand as described in Section 4.2.1 (p. 11).
- The database server is not running or has crashed. Check that it is active and start it if not.
- The database server is not reachable (temporarily or permanently) from the computer the Meal Planner server is running on. You may want to check whether you can connect to the Internet and ensure that your network settings are correct.

4.3 The "MPAjax error" message box

If the dialogue box shown in Figure 4.6 (p. 14) appears, it means that your web browser was unable to communicate with the server. The reason for this could be that:

- the server has not been started. Check that the server is running, and start if it is not.
- the server has unexpectedly crashed. Again, check that that server is still running.
- the computer the server is running on is not reachable (temporarily or permanently) from the computer the browser is running on. You may want to check whether you can connect to the Internet and ensure that your network settings are correct.

There may be other causes that are not listed here. If you are still unsure what the cause of the problem could be, cosider filing a support ticket (see Section 6 (p. 15)). Most modern browsers have what is called a *debug console*; more information about the cause of the error may be found here. It is recommended to include this information with your support ticket. The debug console can be opened, depending on your browser, as follows:

Microsoft Internet Explorer Open the menu by pressing the button with the icon of a gear at the top right of your browser and select the "Developer Tools" item from the menu, as shown in Figure 4.7. The keyboard shortcut for this is usually the F12 key.

Once the Developer Tools panel loads, press the button circled in Figure 4.8 to display the debug console. Right-click the main area of the panel and select "Copy all", as depicted in Figure 4.9, to copy the text to your clipboard so that you can then paste it into your support ticket.

Google Chrome Open the menu by pressing the button at the top right of your btowser and select the "JavaScript Console" item of the "Tools" submenu, as shown in Figure 4.10. The keyboard shortcut for this is usually Ctrl+Shift+J.

Once the panel loads, copy and paste the text in the main body of the panel into your support ticket.

Mozilla Firefox Open the menu by pressing the button labelled "Firefox" at the top left of your browser and select the "Web Console" item of the "Web Developer" submenu, as shown in Figure 4.11 (p. 15). The keyboard shortcut for this is usually Ctrl+Shift+K.

Once the panel loads, copy and paste the text in the main body of the panel into your support ticket.

Other browsers A guide on how to open the debug console can often be found by typing "(name of your browser) open javascript console" into a search engine.

If there is no text in the debug console after opening it, you may need to repeat the action that triggered the error whilst the debug console is open for the error information to be recorded.

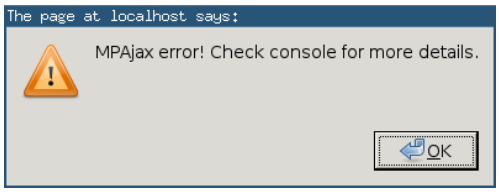

Figure 4.6 – "MPAjax error" dialogue box

5 HTTP error pages

The part of the application that runs in your browser is designed to detect erroneous situations before they reach the server; however in a few cases this is not the case, either due to it being impossible to detect in the browser if the situation is erroneous, or to there

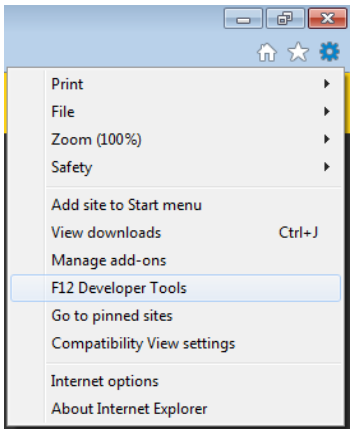

Figure 4.7 – Menu item to open Developer Tools in Internet Explorer

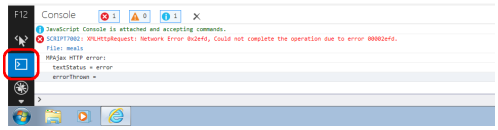

Figure 4.8 – Button to open debug console within Developer Tools in Internet Explorer

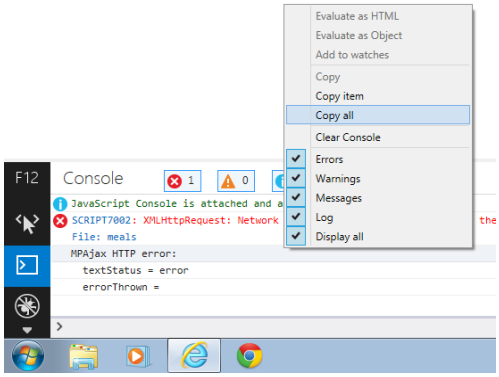

Figure 4.9 – Menu item to copy all debug console text to clipboard in Internet Explorer

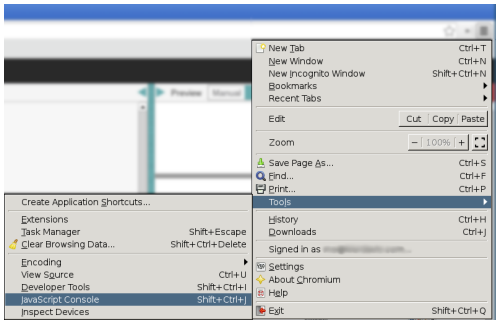

Figure 4.10 – Menu item to open the debug console in Google Chrome

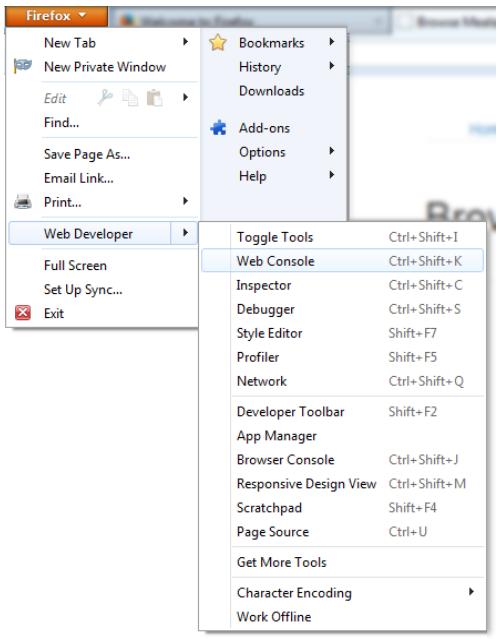

Figure 4.11 – Menu item to open the debug console in Mozilla Firefox

being bugs in the code. If this occurs the server will respond with an "Error" webpage. The different types of these error pages will be listed in this section.

5.1 "Bad Request"

The "Bad Request" error, depicted in Figure 5.1 (p. 16), is caused by invalid input being caught by the server-side validation mechanism, but not the client-side validation. This is usually due to a flaw in the software.

If you see this error page, press your browser's Back button and check that you have entered your information into the form correctly.

5.2 "Not Found"

The "Not Found" error is caused by your browser requesting a page that does not exist. During normal usage of the application this should not happen; however it may occur if, for example, you have bookmarked a meal or meal plan in your browser and then deleted the meal or meal plan, leaving the bookmark pointing to a page that no longer exists.

Note that deleting and then re-creating a meal or meal plan will change its URL and so existing links to it will be invalidated.

If you see this error page, return to the Meal Planner homepage and navigate to the page you were looking for using the browsing and searching pages provided.

5.3 "Internal Server Error"

The "Internal Server Error" is caused by the occurrence of a situation that the server could not recover from. Often this is the result of an error being returned when attempting to access the database.

If you see this error page, check that the database server is running and is accessible from the computer the Meal Planner server is running on.

6 Support

If you experience a problem that is outside the scope of the Troubleshooting section of this manual, find a bug in the application or wish to suggest an enhancement, you should consider creating a support ticket on the project's issue tracker.[4]. You may need to create an account on Github to do this; this is a short procedure.

If your support ticket is describing a problem, you should outline in your ticket:

- what you are trying to do
- what the expected result is
- what actually happens
- your computer's operating system
- the version of the Meal Planner software that you are using

[4]Meal Planner issue tracker: `https://github.com/kierdavis/mealplanner/issues/new`

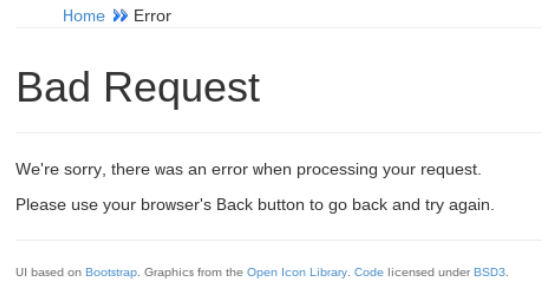

Figure 5.1 – "Bad Request" error page

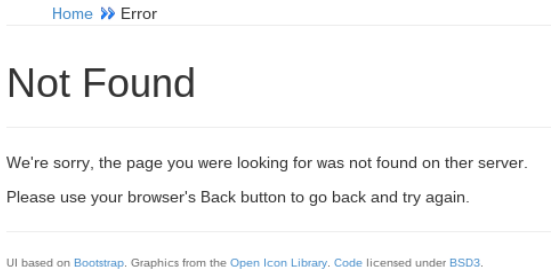

Figure 5.2 – "Not Found" error page

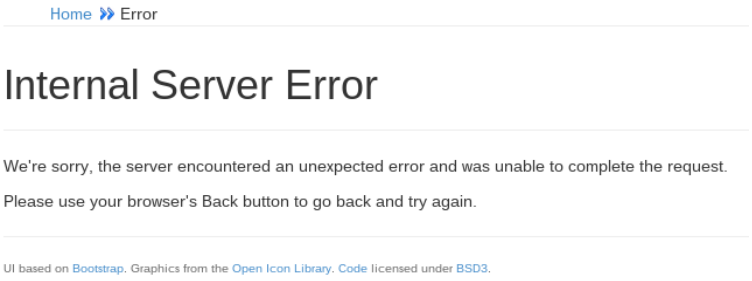

Figure 5.3 – "Internal Server Error" error page

E. System maintenance manual

Meal Planner System Maintenance Manual

Kier Davis

27th November 2015

Contents

List of Figures

List of Tables

List of Code Listings

1 Introduction

The Meal Planner is designed to be a simple yet powerful application to assist in planning meals for the week. It allows you to manage a list of meals, from which suggestions are drawn that you can add to meal plans.

This document will describe the workings of the code and aid in any future maintenance of the application.

It is assumed that you have:

- installed the Git source control system. It can be downloaded from `http://git-scm.com/`.
- installed the Go programming language tools. They can be downloaded from `http://golang.org/`.
- access to a Unix-like shell. On Windows systems, Cygwin[1] or Git Bash (provided in the Git download) will suffice.

[1] `http://www.cygwin.com/`

2 Obtaining the source code

The source code to the application is available on Github. It can be downloaded and built by running the following command:

```
go get -v github.com/kierdavis/mealplanner
```

The source code can now be found under your `GOPATH`[2] source tree, the structure of which is shown in Listing 2.1.

```
bin/
  mealplanner
pkg/
  <os>_<arch>/
    github.com/kierdavis/mealplanner/
      mpapi.a
      mpdata.a
      mpdb.a
      mphandlers.a
      mpresources.a
src/
  github.com/kierdavis/mealplanner/
    mpapi/
    mpdata/
    mpdb/
    mphandlers/
    mpresources/
      resources/
        static/
        templates/
```

Listing 2.1 – Structure of Go workspace

The `bin` folder contains compiled executables. This is where `mealplanner` (Linux) / `mealplanner.exe` (Windows) is placed. This folder should be added to your `PATH`.

The `pkg` folder contains compiled package files. They are organised by the name of your OS and CPU architecture, and the name of the package.

The `src` folder contains package source code. It is organised by the name of the package, meaning that the Meal Planner source code can be found in `$GOPATH/src/github.com/kierdavis/mealplanner`. This folder contains a sub-directory for each sub-package in the project (`mpapi`, `mpdata`, `mpdb`, `mphandlers` and `mpresources`).

[2] The `GOPATH` is the environment variable pointing to a workspace in which your Go source code and build artifacts will be placed. You should have set this up after installing the Go tools. See `http://golang.org/doc/code.html` for more information.

3 Packages

The source is divided into six packages. They are listed as follows:

`github.com/kierdavis/mealplanner` The main server program. It initialises the database and then runs the HTTP server using the handlers defined in `mphandlers`.

`github.com/kierdavis/mealplanner/mpapi` Contains handlers for various API calls, as well as an HTTP handler to dispatch calls to the appropriate API call handler.

`github.com/kierdavis/mealplanner/mpdata` Contains definitions of the data types and scoring algorithm.

`github.com/kierdavis/mealplanner/mpdb` Contains routines for accessing the database in consistent manner.

`github.com/kierdavis/mealplanner/mphandlers` Contains handlers for various HTTP requests.

`github.com/kierdavis/mealplanner/mpresources` Contains the HTML templates and static files, and functions to access these on startup.

A list of external dependencies of the project is given below:

`github.com/go-sql-driver/mysql` A package implementing the standard interface for SQL databases (defined in the `database/sql` package) using MySQL as a backend. This package is used for low-level access to a database, on top of which the `mpdb` abstraction layer is built.

`github.com/gorilla/mux` A package that provides more advanced request routing capabilities than is present in the standard `net/http` package, allowing request handlers to be chosen based on the request method, HTTP headers and form fields as well as the request URL. It is part of the Gorilla[3] web toolkit.

4 Structure of the code

Figure 4.1 shows the various server and client modules in the application, and the links between them. It can be seen that:

- `mpdb` acts as an abstraction layer between the low-level `database/sql` package and the HTTP and API request handlers.
- Likewise, `MPAjax` acts as an abstraction layer between jQuery's Ajax layer and the Javascript that controls each webpage.

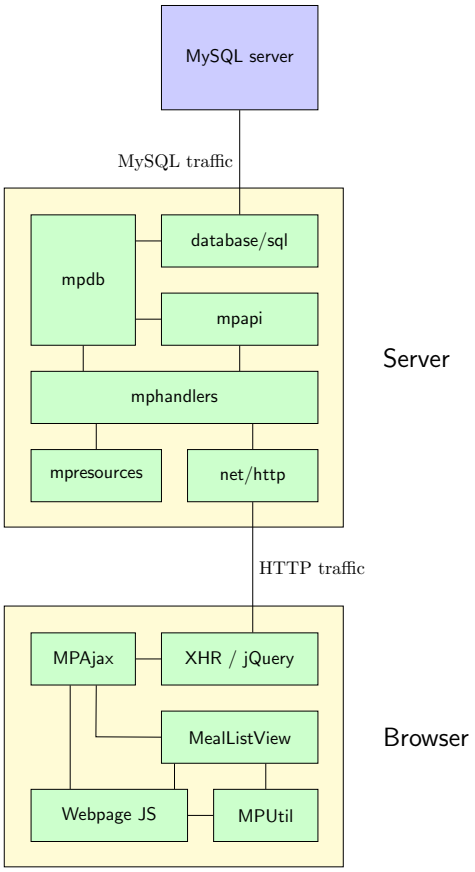

Figure 4.1 – Interconnectivity of system modules

[3] `http://www.gorillatoolkit.org/`

5 Algorithms

The pseudocode in this section:

- uses the syntax a_k to refer to any of
  - the item at index k in the list/array/vector a
  - the value at key k in the map/hash/associative array a
  - the value of field k in the structure/object a
- uses the syntax **for all** $x \in l$ to represent iteration over a list. Each item of the list l is successively bound to the variable x.

5.1 Suggestion generation

The pseudocode for the suggestion generation algorithm is given in Listing 5.1 (p. 6). This algorithm calculates a score for each meal in the database (based on the date of the meal plan day that is being edited), sorts the meals by their score and returns them as a list of suggestions.

In the code, this functionality is distributed between `suggs.go` in the `mpdb` package and `score.go` in the `mpdata` package. `suggs.go` retrieves the necessary information from the database and passes it to `score.go`, which contains the core scoring algorithm.

A description of the functions defined in the pseudocode is as follows:

FindMinServingDist($mealID$**,** $dateBeingEdited$**)** returns the date difference (i.e. the number of days between) $dateBeingEdited$ and the closest serving to $dateBeingEdited$ of the meal identified by $mealID$.

CalcTagScores($dateBeingEdited$**)** calculates a score for each tag in the database and returns these as a map (also called a hash or associative array) of tags to their scores.

ScoreMeal($mealID$**,** $dateBeingEdited$**)** calculates and returns a score for the meal identified by $mealID$ for serving on $dateBeingEdited$.

NormaliseSuggestions($suggs$**)** normalises each score in the list $suggs$ to be between 0 and 1, by subtracting the minimum score from each and then dividing by the range.

GenerateSuggestions($dateBeingEdited$**)** generates a sorted list of suggestions for $dateBeingEdited$.

A description of the external functions used by the functions defined in the pseudocode is as follows:

DateDiff(a**,** b**)** returns the number of days between the dates a and b. If a and b are the same date, 0 is returned; if they are consecutive, 1 is returned and so on.

FetchAllMeals() returns a list of all meal identifiers in the database.

FetchAllTags() returns a list of all $(tag, mealID)$ pairs in the database.

FetchNumServings($mealID$**)** returns the number of servings of the meal identified by $mealID$.

FetchServingsOfMeal($mealID$**)** returns a list of all the dates of the days on which the meal identified by $mealID$ is served.

FetchTagsForMeal($mealID$**)** returns a list of all the tags associated with the meal identified by $mealID$.

First(l**)** returns the first item in the list l.

IsMealFavourite($mealID$**)** returns true if the meal identified by $mealID$ is marked as a favourite, false otherwise.

Last(l**)** returns the last item in the list l.

```
function FindMinServingDist(mealID, dateBeingEdited)
    for all servingDate ∈ FetchServingsOfMeal(mealID) do
        dist ← DateDiff(servingDate, dateBeingEdited)
        if minDist is not set or dist < minDist then
            minDist ← dist
        end if
    end for
    return minDist
end function
function CalcTagScores(dateBeingEdited)
    Allocate a new map, tagScores
    for all (tag, taggedMealID) ∈ FetchAllTags() do
        minDist ← FindMinServingDist(taggedMealID, dateBeingEdited)
        tagScores_tag ← 0.1 + tanh(0.2 × minDist)
    end for
    return tagScores
end function
function ScoreMeal(mealID, dateBeingEdited)
    score ← 1
    if FetchNumServings(mealID) > 0 then
        minDist ← FindMinServingDist(mealID, dateBeingEdited)
        score ← score × (1.45 − (2.8 ÷ (minDist + 1)))
    else
        score ← score × 1.6
    end if
    for all tag ∈ FetchTagsForMeal(mealID) do
        if tagScores_tag exists then
            score ← score × tagScores_tag
        end if
    end for
    if IsMealFavourite(mealID) then
        score ← score × 2
    end if
    return score
end function
function NormaliseSuggestions(suggs)
    maxScore ← First(suggs)_score          ▷ suggs is sorted in descending order, so the highest
    minScore ← Last(suggs)_score           ▷ scoring suggestion is the first and the lowest is the last.
    range ← maxScore − minScore
    for all sugg ∈ suggs do
        sugg_score ← (sugg_score − minScore) ÷ range
    end for
    return suggs
end function
function GenerateSuggestions(dateBeingEdited)
    tagScores ← CalcTagScores(dateBeingEdited)
    Allocate a new list, suggs
    for all mealID ∈ FetchAllMeals() do
        mealScore ← ScoreMeal(mealID, dateBeingEdited)
        Append (mealID, mealScore) to suggs
    end for
    Sort suggs by descending score
    suggs ← NormaliseSuggestons(suggs)
    return suggs
end function
```

Listing 5.1 – Pseudocode for suggestion generation

5.2 Database initialisation

The pseudocode for the database initialisation algorithm is given in Listing 5.1 (p. 8). This algorithm ensures that a version number is set in the database, creates the tables if needed and migrates the database to the latest version.

In the application this algorithm is contained in `tables.go` and `migration.go` in the `mpdb` sub-package, with `migration.go` handling the migration-specific element of the algorithm and `tables.go` performing the rest.

Section 6.2 goes into further detail on the concepts involved in database migration.

A description of the functions defined in the pseudocode is as follows:

InitVersion() checks to see if a version number is stored in the database, and sets one if not.

FindBestMigration($currentVersion$**,** $maxFinishVersion$**)** finds the migration step that starts at $currentVersion$, finishes no later than $maxFinishVersion$ and spans the most versions.

Migrate($targetVersion$**)** applies migration steps to the database until it reaches $targetVersion$.

InitDB() performs the entire initialisation procedure.

A description of the external functions used by the functions defined in the pseudocode is as follows:

ApplyMigration($migration$**)** applies the migration step $migration$ to the database by executing its SQL commands.

CreateTables() executes the SQL commands necessary to create each of the four database tables.

GetAllMigrations() returns the list of all migration steps that could be applied.

GetVersion() returns the version number that is stored in the database.

IsVersionSet() returns true if there is a version number stored in the database, and false otherwise.

MealTableExists() returns true if the `meal` table exists in the database, and false otherwise.

SetVersion($version$**)** sets the version number stored in the database to $version$.

6 Database maintenance

6.1 Schemas

The database schemas (for database version 1) are given in Listing 6.1.

```
CREATE TABLE IF NOT EXISTS meal (
  id BIGINT UNSIGNED NOT NULL
      AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  recipe TEXT,
  favourite BOOLEAN NOT NULL,
  searchtext TEXT NOT NULL,
  PRIMARY KEY (id)
);
CREATE TABLE IF NOT EXISTS tag (
  mealid BIGINT UNSIGNED NOT NULL,
  tag VARCHAR(64) NOT NULL,
  PRIMARY KEY (mealid, tag)
);
CREATE TABLE IF NOT EXISTS mealplan (
  id BIGINT UNSIGNED NOT NULL
      AUTO_INCREMENT,
  notes TEXT,
  startdate DATE NOT NULL,
  enddate DATE NOT NULL,
  PRIMARY KEY (id)
);
CREATE TABLE IF NOT EXISTS serving (
  mealplanid BIGINT UNSIGNED NOT NULL,
  dateserved DATE NOT NULL,
  mealid BIGINT UNSIGNED NOT NULL,
  PRIMARY KEY (mealplanid, dateserved)
);
```

Listing 6.1 – Database schemas

6.2 Migration and versioning

The application contains a database migration system, allowing changes to the database structure to be made whilst maintaining backwards compatibility. It works by storing a version number in every database, and comparing it to the latest version upon startup. If it is behind the latest version then migration steps are applied to bring the database up to date. Newly created databases use the latest version.

A list of database versions, as of application version 1.0.1, is given in Table 6.1.

Version	Notes
0	First version
1	Addition of `searchtext` column

Table 6.1 – List of database version (as of application version 1.0.1)

6.2.1 Making a change to the database structure

The addition of the `searchtext` column featured in database version 1 will be used as an example in the following guide on how to make a change to the database structure:

```
function InitVersion
    if not IsVersionSet() then
        if MealTableExists() then                ▷ Meal table exists but not version table, assume first
            SetVersion(0)                        ▷ server startup since introduction of versioning.
        else                                     ▷ Neither meal table nor version table
            SetVersion(latestVersion)            ▷ exist, assume empty database
        end if
    end if
end function
function FindBestMigration(currentVersion, maxFinishVersion)
    for all migration ∈ GetAllMigrations() do
        if migration_start == currentVersion and migration_finish <= maxFinishVersion then
            if bestMigration is unset or migration_finish > bestMigration_finish then
                bestMigration ← migration
            end if
        end if
    end for
    return bestMigration
end function
function Migrate(targetVersion)
    currentVersion ← GetVersion()
    if currentVersion > targetVersion then
        return error
    end if
    while currentVersion < targetVersion do
        migration ← FindBestMigration(currentVersion, targetVersion)
        ApplyMigration(migration)
        currentVersion ← migration_finish
    end while
    SetVersion(targetVersion)
end function
function InitDB
    InitVersion()
    CreateTables()
    Migrate(latestVersion)
end function
```

Figure 5.1 – Pseudocode for database initalisation

1. Increment the `LatestVersion` constant in `migration.go`, in the `mpdb` package.
2. Determine the SQL commands that will perform the change. For the example, these are given in Listing 6.2.
3. Add a new item to the `Migrations` variable in `migration.go`. The `Migration` struct has three fields: the starting version, the finishing version and the list of SQL commands to do. Assuming the latest database version is 1 this definition is given in Listing 6.3.
4. Modify the existing code to reflect the new change. Bear in mind that it is ensured that the database is at the latest version before any of the main application code, including table creation, is executed. In the case of the example, the table creation SQL is updated to add a `searchtext` column.

```
ALTER TABLE meal ADD COLUMN searchtext
    TEXT NOT NULL;
UPDATE meal SET meal.searchtext = CONCAT
    (meal.name, ' ', meal.recipe);
```

Listing 6.2 – Commands to perform exemplar database change

```
&Migration{1, 2, []string{
  "ALTER TABLE meal ADD COLUMN
      searchtext TEXT NOT NULL",
  "UPDATE meal SET meal.searchtext =
      CONCAT(meal.name, ' ', meal.recipe
      )",
}},
```

Listing 6.3 – Go code to create a `Migration` struct representing the exemplar database change

Appendices

A Godoc API documentation

API documentation for the public interface to all five packages is available online at `http://godoc.org/github.com/kierdavis/mealplanner`. It is also reproduced here.

package mpapi

import "github.com/kierdavis/mealplanner/mpapi"

Index

Package Files

deletemeal.go deleteserving.go fetchalltags.go fetchmeallist.go fetchmealplans.go fetchservings.go fetchsuggestions.go mpapi.go togglefavourite.go updatenotes.go updateserving.go

func HandleAPICall

```
func HandleAPICall(w http.ResponseWriter, r *http.Request)
```

HandleAPICall handles an HTTP request for an API call. It obtains the form values, passes them through Dispatch and sends the resulting JSON response to the client.

type JSONResponse

```
type JSONResponse struct {
    Error   string      `json:"error"`   // The error message, in the event of an unsuccessful response.
    Success interface{} `json:"success"` // The response payload, in the event of a successful response.
}
```

JSONResponse contains the response structure returned to the client. If the 'Error' field is nonempty, the response indicates an error has occurred, else the response is assumed to be a successful one.

func Dispatch

```
func Dispatch(params url.Values) (response JSONResponse)
```

Dispatch inspects the "command" parameter and dispatches the request to the appropriate handler function.

Package mpapi imports 10 packages (graph) and is imported by 1 packages. Updated about a minute ago. Refresh now. Tools for package owners.

GoDoc

mealplanner: github.com/kierdavis/mealplanner/mpdata Index | Files

package mpdata

import "github.com/kierdavis/mealplanner/mpdata"

Package mpdata defines the data model and core algorithms used by the application.

Index

Package Files

mealplanjson.go mpdata.go score.go suggestionslice.go types.go

Constants

```
const DatepickerDateFormat = "02/01/2006"
```

DatepickerDateFormat is the time format used by the JQuery datepicker widget and sent in POST requests. See also: documentation on 'time.Parse'.

```
const JSONDateFormat = "2006-01-02"
```

JSONDateFormat is the time format used in JSON encodings. See also: documentation on 'time.Parse'.

type Meal

```
type Meal struct {
    ID        uint64 `json:"id"`        // The database's unique identifier for the meal.
    Name      string `json:"name"`      // The name of the meal.
    RecipeURL string `json:"recipe"`    // The possibly empty URL of the recipe for the meal.
    Favourite bool   `json:"favourite"` // Whether or not the meal is marked as a favourite.
}
```

Meal holds information about a meal in the database.

type MealPlan

```
type MealPlan struct {
    ID        uint64    // The database's unique identifier for the meal plan.
    Notes     string    // The textual notes associated with the meal plan.
    StartDate time.Time // The date of the first day in the meal plan.
    EndDate   time.Time // The date of the last day in the meal plan.
}
```

MealPlan holds information about a meal plan in the database. It contains no JSON field tags as the mealPlanJSON struct is actually used for encoding/decoding; however, the MarshalJSON/UnmarshalJSON methods take care of this.

func (*MealPlan) Days

```
func (mp *MealPlan) Days() (days []time.Time)
```

Days returns a slice of times representing the days between mp.StartDate and mp.EndDate, inclusive.

func (*MealPlan) MarshalJSON

```
func (mp *MealPlan) MarshalJSON() (text []byte, err error)
```

MarshalJSON encodes a meal plan into its JSON form.

func (*MealPlan) UnmarshalJSON

```
func (mp *MealPlan) UnmarshalJSON(text []byte) (err error)
```

UnmarshalJSON populates the fields of the receiver with values parsed from the input JSON.

type MealPlanWithServings

```
type MealPlanWithServings struct {
    MealPlan *MealPlan  `json:"mealplan"`
    Servings []*Serving `json:"servings"`
}
```

MealPlanWithServings pairs a MealPlan with its associated Servings.

type MealWithTags

```
type MealWithTags struct {
    Meal *Meal    `json:"meal"` // The meal.
    Tags []string `json:"tags"` // The meal's tags.
}
```

MealWithTags pairs a Meal with its associated tags.

type Scorer

```
type Scorer struct {
    // contains filtered or unexported fields
}
```

Scorer encapsulates the scoring algorithm used for suggestion generation.

func NewScorer

```
func NewScorer() (s *Scorer)
```

NewScorer allocates and returns a new Scorer.

func (*Scorer) AddTagScore

```
func (s *Scorer) AddTagScore(tag string, dist int)
```

AddTagScore should be called for every tag occurrence in the database. The 'dist' argument refers to the number of days between the date that suggestions are being generated for and the date of the closest serving of the meal the tag is associated with.

func (*Scorer) ScoreSuggestion

```
func (s *Scorer) ScoreSuggestion(sugg *Suggestion)
```

ScoreSuggestion calculates a score for the given suggestion and assigns it to the 'Score' field of the argument.

type Serving

```
type Serving struct {
    MealPlanID uint64
    Date       time.Time
    MealID     uint64
}
```

Serving holds information about a serving of a meal in the database. It contains no JSON field tags as the servingJSON struct is actually used for encoding/decoding; however, the MarshalJSON/UnmarshalJSON methods take care of this.

func (*Serving) MarshalJSON

```
func (s *Serving) MarshalJSON() (text []byte, err error)
```

MarshalJSON encodes a meal serving into its JSON form.

func (*Serving) UnmarshalJSON

```
func (s *Serving) UnmarshalJSON(text []byte) (err error)
```

UnmarshalJSON populates the fields of the receiver with values parsed from the input JSON.

type Suggestion

```
type Suggestion struct {
    MT    MealWithTags `json:"mt"`    // The meal and tags.
    CSD   int          `json:"-"`     // The closest serving distance (used in computing the score).
    Score float32      `json:"score"` // The meal's score.
}
```

Suggestion pairs a Meal with its associated tags, closest serving distance and score.

type SuggestionSlice

```
type SuggestionSlice []*Suggestion
```

SuggestionSlice is an implementation of 'sort.Interface', allowing a list of suggestions to be sorted by their score.

func (SuggestionSlice) Len

```
func (ss SuggestionSlice) Len() (n int)
```

Len returns the number of suggestions in the list.

func (SuggestionSlice) Less

```
func (ss SuggestionSlice) Less(i int, j int) (less bool)
```

Less returns whether the suggestion indexed by 'i' has a higher score than the suggestion indexed by 'j' and so should be placed closer to the start of the list.

func (SuggestionSlice) Swap

```
func (ss SuggestionSlice) Swap(i int, j int)
```

Swap exchanges the suggestions indexed by 'i' and 'j'.

GoDoc

mealplanner: github.com/kierdavis/mealplanner/mpdb Index | Files

package mpdb

import "github.com/kierdavis/mealplanner/mpdb"

Package mpdb provides routines for manipulating the database whilst preserving referential integrity as best as possible.

Index

Package Files

meal.go mealplan.go migration.go mpdb.go suggs.go tables.go

Constants

```
const DBDriver = "mysql"
```

DBDriver is the driver name used when connecting to the database.

```
const DBParams = "?parseTime=true"
```

DBParams are extra parameters required for the database routines to function.

```
const LatestVersion = 1
```

LatestVersion is the latest version a database could be in.

```
const SearchTextExpr = "CONCAT(meal.name, ' ', meal.recipe, ' ', IFNULL((SELECT GROUP_CONCAT(tag.tag S
```

SQL expression to find the contents of the "searchtext" field for a meal.

Variables

```
var DBSource = "mealplanner@unix(/var/run/mysqld/mysqld.sock)/mealplanner"
```

DBSource identifies how to connect to the database. It should take the form "USER:PASS@unix(/PATH/TO/SOCKET)/DBNAME" or "USER:PASS@tcp(HOST:PORT)/DBNAME". By default, it will attempt to connect via the local Unix socket to the 'mealplanner' database, with username 'mealplanner' and no password.

```
var Migrations = []*Migration{

    &Migration{0, 1, []string{
        "ALTER TABLE meal ADD COLUMN searchtext TEXT NOT NULL",
        "UPDATE meal SET meal.searchtext = " + SearchTextExpr,
    }},
}
```

Migrations contains a list of all possible migration steps.

func AddMeal

```
func AddMeal(q Queryable, meal *mpdata.Meal) (err error)
```

AddMeal adds the information in 'meal' to the database as a new record, then sets 'meal.ID' to the identifier of this new record.

func AddMealPlan

```
func AddMealPlan(q Queryable, mp *mpdata.MealPlan) (err error)
```

AddMealPlan adds the information contained in 'mp' to the database as a new meal plan record. It assigns the identifier of the newly created record to the ID field of the meal plan.

func AddMealTags

```
func AddMealTags(q Queryable, mealID uint64, tags []string) (err error)
```

AddMealTags adds the the list of tags given in 'tags' to the meal identified by 'mealID'.

func AddMealWithTags

```
func AddMealWithTags(q Queryable, mt mpdata.MealWithTags) (err error)
```

AddMealWithTags combines 'AddMeal' and 'AddMealTags'.

func AddServing

```
func AddServing(q Queryable, serving *mpdata.Serving) (err error)
```

AddServing adds the information containing in 'serving' to a new serving record in the database.

func AttachMealTags

```
func AttachMealTags(q Queryable, meals []*mpdata.Meal) (mts []mpdata.MealWithTags, err error)
```

AttachMealTags takes a list of meals, looks up the tags for each one and returns the results.

func AutoFillMealPlan

```
func AutoFillMealPlan(q Queryable, mp *mpdata.MealPlan) (err error)
```

AutoFillMealPlan assigns servings to every day in 'mp' using the top suggestion for each day.

func AutoFillMealPlanDay

```
func AutoFillMealPlanDay(q Queryable, mpID uint64, date time.Time) (err error)
```

AutoFillMealPlanDay assigns a serving to day 'date' on the meal plan identified by 'mpID' using the top suggestion.

func ClearTables

```
func ClearTables(q Queryable) (err error)
```

ClearTables deletes all records from the entire database.

func Connect

```
func Connect() (db *sql.DB, err error)
```

Connect creates a new connection to the database using DBDriver and DB_SOURCE.

func CountServings

```
func CountServings(q Queryable, mpID uint64) (numServings int, err error)
```

CountServings returns the number of servings in the meal plan identified by 'mpID'.

func CreateTables

```
func CreateTables(q Queryable) (err error)
```

CreateTables creates the database tables if they do not exist.

func DeleteMeal

```
func DeleteMeal(q Queryable, mealID uint64) (err error)
```

DeleteMeal deletes the meal record identified by 'mealID'. If no such meal exists, no error is raised.

func DeleteMealPlan

```
func DeleteMealPlan(q Queryable, mpID uint64) (err error)
```

DeleteMealPlan deletes the meal plan record identified by 'mpID'. If no such meal plan exists, no error is raised.

func DeleteMealTags

```
func DeleteMealTags(q Queryable, mealID uint64) (err error)
```

DeleteMealTags deletes all tags in the database associated with the meal identified by 'mealID'. If no such tags exist, no error is raised.

func DeleteMealWithTags

```
func DeleteMealWithTags(q Queryable, mealID uint64) (err error)
```

DeleteMealWithTags deletes the meal record identified by 'mealID', and all tag records associated with it.

func DeleteServing

```
func DeleteServing(q Queryable, mpID uint64, date time.Time) (err error)
```

DeleteServing deletes the serving at 'date' on the meal plan identified by 'mpID'. If no such serving exists, no error is raised.

func DeleteServings

```
func DeleteServings(q Queryable, mpID uint64) (err error)
```

DeleteServings deletes all servings on the meal plan identified by 'mpID'. If no such servings exist, no error is raised.

func DeleteServingsOf

```
func DeleteServingsOf(q Queryable, mealID uint64) (err error)
```

DeleteServingsOf deletes all servings of the meal identified by 'mealID'. IF no such servings exist, no error is raised.

func DeleteTables

```
func DeleteTables(q Queryable) (err error)
```

DeleteTables drops the database tables if they exist.

func GenerateSuggestions

```
func GenerateSuggestions(q Queryable, mpID uint64, date time.Time) (suggs []*mpdata.Suggestion, err error)
```

GenerateSuggestions calculates a score for each meal in the database based on their suitability for serving on 'date'. These are returned as a list of Suggestions.

func GetDatabaseVersion

```
func GetDatabaseVersion(q Queryable) (v uint, err error)
```

GetDatabaseVersion fetches and returns the version number of the database.

func GetMeal

```
func GetMeal(q Queryable, mealID uint64) (meal *mpdata.Meal, err error)
```

GetMeal fetches information from the database about the meal identified by 'mealID'.

func GetMealPlan

```
func GetMealPlan(q Queryable, mpID uint64) (mp *mpdata.MealPlan, err error)
```

GetMealPlan returns information about the meal plan identified by 'mpID'.

func GetMealPlanWithServings

```
func GetMealPlanWithServings(q Queryable, mpID uint64) (mps *mpdata.MealPlanWithServings, err error)
```

GetMealPlanWithServings returns the information about the meal plan identified by 'mpID' including its servings.

func GetMealTags

```
func GetMealTags(q Queryable, mealID uint64) (tags []string, err error)
```

GetMealTags fetches the list of tags associated with the meal identified by 'mealID'.

func GetMealWithTags

```
func GetMealWithTags(q Queryable, mealID uint64) (mt mpdata.MealWithTags, err error)
```

GetMealWithTags combines GetMeal and GetMealTags.

func GetServing

```
func GetServing(q Queryable, mpID uint64, date time.Time) (serving *mpdata.Serving, err error)
```

GetServing returns information about the meal serving identified by the meal plan identifier 'mpID' and the serving date 'date'.

func GetServings

```
func GetServings(q Queryable, mpID uint64) (servings []*mpdata.Serving, err error)
```

GetServings returns a slice containing the servings that are part of the meal plan identified by 'mpID'.

func InitDB

```
func InitDB(debug bool, testData bool) (err error)
```

InitDB creates the database tables if they don't exist. If 'debug' is true, debug messages are printed. If 'testData' is true, the tables are also cleared and test data are added to them.

func InitialiseVersion

```
func InitialiseVersion(q Queryable, debug bool) (err error)
```

InitialiseVersion ensures that the database version is set. If it is not present, it is decided whether it is an empty database or the first startup since the introduction of versioning. The database can now be safely migrated to the latest version after calling this function.

func InsertTestData

```
func InsertTestData(q Queryable) (err error)
```

InsertTestData inserts some predefined meals and meal plans into the database for testing purposes.

func ListAllTags

```
func ListAllTags(q Queryable, sortByName bool) (tags []string, err error)
```

ListAllTags returns a list (without duplicates) of all tags that appear in the database. If the 'sortByName' parameter is true, the tags are sorted into alphabetical order.

func ListMealPlansBetween

```
func ListMealPlansBetween(q Queryable, from time.Time, to time.Time) (mps []*mpdata.MealPlan, err error)
```

ListMealPlansBetween returns a list of all meal plans in the database whose date range (start date to end date) overlaps with the given date range ('from' to 'to').

func ListMeals

```
func ListMeals(q Queryable, sortByName bool) (meals []*mpdata.Meal, err error)
```

ListMeals fetches and returns a list of all meals in the database. If the parameter 'sortByName' is true, the meals are sorted in alphabetical order by name.

func ListMealsWithTags

```
func ListMealsWithTags(q Queryable, sortByName bool) (mts []mpdata.MealWithTags, err error)
```

ListMealsWithTags fetches and returns a list of all meals in the database with their associated tags. If the parameter 'sortByName' is true, the meals are sorted in alphabetical order by name.

func Migrate

```
func Migrate(q Queryable, targetVersion uint, debug bool) (err error)
```

Migrate migrates the database from the current version to 'targetVersion'. If 'debug' is true, messages are printed to stdout describing the operations taking place.

func SearchMeals

```
func SearchMeals(q Queryable, words []string, sortByName bool) (meals []*mpdata.Meal, err error)
```

SearchMeals fetches and returns a list of all meals in the database where all of the strings given in 'words' are included somewhere in the meal's searchtext. If the parameter 'sortByName' is true, the meals are sorted in alphabetical order by name.

func SearchMealsWithTags

```
func SearchMealsWithTags(q Queryable, words []string, sortByName bool) (mts []mpdata.MealWithTags, err error)
```

SearchMealsWithTags fetches and returns a list of all meals in the database - with their associated tags - where all of the strings given in 'words' are included somewhere in the meal's searchtext. If the parameter 'sortByName' is true, the meals are sorted in alphabetical order by name.

func SetDatabaseVersion

```
func SetDatabaseVersion(q Queryable, v uint) (err error)
```

SetDatabaseVersion updates the version number in the database.

func ToggleFavourite

```
func ToggleFavourite(q Queryable, mealID uint64) (isFavourite bool, err error)
```

ToggleFavourite toggles the "favourite" status of the meal identified by 'mealID', and returns the new favourite status.

func UpdateMeal

```
func UpdateMeal(q Queryable, meal *mpdata.Meal) (err error)
```

UpdateMeal replaces with the information in the database for the meal identified by 'meal.ID' with the information in 'meal'.

func UpdateMealTags

```
func UpdateMealTags(q Queryable, mealID uint64, tags []string) (err error)
```

UpdateMealTags replaces the tags associated with the meal identified by 'mealID' with the list given by 'tags'.

func UpdateMealWithTags

```
func UpdateMealWithTags(q Queryable, mt mpdata.MealWithTags) (err error)
```

UpdateMealWithTags combines UpdateMeal and UpdateMealTags.

func UpdateNotes

```
func UpdateNotes(q Queryable, mpID uint64, notes string) (err error)
```

UpdateNotes sets the notes associated with the meal plan identified by 'mpID' to 'notes'.

func UpdateSearchText

```
func UpdateSearchText(q Queryable, mealID uint64) (err error)
```

UpdateSearchText sets the searchtext of a meal based on its name, recipe URL and tags.

func WithConnection

```
func WithConnection(f WithConnectionFunc) (err error)
```

WithConnection opens a connection to the database, calls 'f' with the database as a parameter, then ensures the database is closed even in the event of an error. If an error occurs when closing the database, a 'FailedCloseError' is returned.

func WithTransaction

```
func WithTransaction(db *sql.DB, f WithTransactionFunc) (err error)
```

WithTransaction begins a transaction on the given database connection, calls 'f' with the transaction as a parameter, then ensures the transaction is committed if 'f' completes successfully or rolled back in the event of an error. If an error occurs when committing or rolling back the transaction, a 'FailedCloseError' is returned.

type FailedCloseError

```
type FailedCloseError struct {
    What          string // A string used in the error message to identify what resource was being closed.
    CloseError    error  // The error returned when the resource was closed.
    OriginalError error  // The original error that triggered the closing of the resource.
}
```

FailedCloseError contains information regarding a situation where an error occurs when closing a resource in response to an earlier error.

func (*FailedCloseError) Error

```
func (err *FailedCloseError) Error() (msg string)
```

Error formats the information contained in 'err' into an error message.

type LoggingQueryable

```
type LoggingQueryable struct {
    Q Queryable
}
```

LoggingQueryable wraps a Queryable while logging all executions of its functions to standard output. It is intended for debugging purposes.

func (LoggingQueryable) Exec

```
func (lq LoggingQueryable) Exec(query string, args ...interface{}) (result sql.Result, err error)
```

Exec executes a query without returning any rows. The args are for any placeholder parameters in the query.

func (LoggingQueryable) Prepare

```
func (lq LoggingQueryable) Prepare(query string) (stmt *sql.Stmt, err error)
```

Prepare creates a prepared statement for later queries or executions. Multiple queries or executions may be run concurrently from the returned statement.

func (LoggingQueryable) Query

```
func (lq LoggingQueryable) Query(query string, args ...interface{}) (rows *sql.Rows, err error)
```

Query executes a query that returns rows, typically a SELECT. The args are for any placeholder parameters in the query.

func (LoggingQueryable) QueryRow

```
func (lq LoggingQueryable) QueryRow(query string, args ...interface{}) (row *sql.Row)
```

QueryRow executes a query that is expected to return at most one row. QueryRow always return a non-

nil value. Errors are deferred until Row's Scan method is called.

type Migration

```
type Migration struct {
    From  uint     // The version the database must be at before this migration is executed.
    To    uint     // The version the database will be at after this migration is executed.
    Stmts []string // The SQL statements that perform the migration.
}
```

A Migration represents a possible migration step between two database versions.

func FindMigration

```
func FindMigration(from uint, maxTo uint) (m *Migration)
```

FindMigration finds a migration step that starts at version 'from' and finishes anywhere up to and including version 'maxTo'. If there are multiple possible choices, the one that spans the most versions (i.e. the one with the highest "to" version) is returned.

func (*Migration) Apply

```
func (m *Migration) Apply(q Queryable) (err error)
```

Apply runs the migration against a database.

type MigrationError

```
type MigrationError struct {
    From    uint   // The version we were attempting to migrate from.
    To      uint   // The version we were attempting to migrate to.
    Message string // The error message.
}
```

A MigrationError is returned when database migration failed.

func (MigrationError) Error

```
func (e MigrationError) Error() (msg string)
```

type Queryable

```
type Queryable interface {
    Exec(string, ...interface{}) (sql.Result, error)
    Prepare(string) (*sql.Stmt, error)
    Query(string, ...interface{}) (*sql.Rows, error)
    QueryRow(string, ...interface{}) *sql.Row
}
```

Queryable represents a type that can be queried (either a *sql.DB or *sql.Tx). See documentation on 'database/sql#DB' for information on the methods in this interface.

type WithConnectionFunc

```
type WithConnectionFunc func(*sql.DB) error
```

WithConnectionFunc represents a function that can be used with WithConnection.

type WithTransactionFunc

```
type WithTransactionFunc func(*sql.Tx) error
```

WithTransactionFunc represents a function that can be used with WithTransaction.

Package mpdb imports 7 packages (graph) and is imported by 3 packages. Updated 2014-03-20. Refresh now. Tools for package owners.

GoDoc

mealplanner: github.com/kierdavis/mealplanner/mphandlers Index | Files

package mphandlers

import "github.com/kierdavis/mealplanner/mphandlers"

Package mphandlers defines the HTTP handlers for the application.

Index

Package Files

addmeal.go browsemealplans.go browsemeals.go createmealplan.go deletemealplan.go editmeal.go editmealplan.go home.go httperror.go logging.go mphandlers.go util.go viewmealplan.go

Variables

```
var BadRequestError = &HTTPError{
    Status:    http.StatusBadRequest,
    ShortDesc: "Bad Request",
    LongDesc:  "We're sorry, there was an error when processing your request.",
}
```

BadRequestError represents an HTTP 400 Bad Request error.

```
var InternalServerError = &HTTPError{
    Status:    http.StatusInternalServerError,
    ShortDesc: "Internal Server Error",
    LongDesc:  "We're sorry, the server encountered an unexpected error and was unable to complete the request."
}
```

InternalServerError represents an HTTP 500 Internal Server Error.

```
var NotFoundError = &HTTPError{
    Status:    http.StatusNotFound,
    ShortDesc: "Not Found",
    LongDesc:  "We're sorry, the page you were looking for was not found on ther server.",
}
```

NotFoundError represents an HTTP 404 Not Found error.

func CreateMux

```
func CreateMux() (m *mux.Router)
```

CreateMux creates a *mux.Router and attaches the application's HTTP handlers to it.

type HTTPError

```
type HTTPError struct {
    Status    int    // The HTTP status code.
    ShortDesc string // The associated "reason" message sent with the status code.
    LongDesc  string // A longer message displayed to the user on the HTML error page.
}
```

HTTPError holds related information about an HTTP status code used by the application.

type LoggingHandler

```
type LoggingHandler struct {
    Handler http.Handler
}
```

LoggingHandler wraps an http.Handler, printing a message to standard output whenever a request is handled.

func (LoggingHandler) ServeHTTP

```
func (lh LoggingHandler) ServeHTTP(w http.ResponseWriter, r *http.Request)
```

ServeHTTP handles an HTTP request.

Package mphandlers imports 12 packages (graph) and is imported by 1 packages. Updated about a minute ago. Refresh now. Tools for package owners.

GoDoc

mealplanner: github.com/kierdavis/mealplanner/mpresources Index | Files

package mpresources

import "github.com/kierdavis/mealplanner/mpresources"

Package mpresources contains the HTML templates and static files used by the application.

Index

Package Files

mpresources.go

func GetResourceDir

```
func GetResourceDir() (dir string)
```

GetResourceDir returns the resource directory. If it is uninitialised, it looks for the package's source directory in the GOPATH and uses that.

func GetStaticDir

```
func GetStaticDir() (dir string)
```

GetStaticDir returns the directory used for storing static files.

func GetTemplates

```
func GetTemplates() (t *template.Template)
```

GetTemplates loads the templates from the resource directory if they have not been loaded already, and returns them.

func SetResourceDir

```
func SetResourceDir(dir string)
```

SetResourceDir sets the resource directory.

Package mpresources imports 3 packages (graph) and is imported by 2 packages. Updated about a minute ago. Refresh now. Tools for package owners.

F. Code listings

F.1 Server-side code

F.1.1 Listing of mealplanner.go

```
// Command mealplanner is the main entry point of the application. It simply
// runs the *mux.Router provided by mphandlers.CreateMux() as an HTTP server.
package main
import (
    "flag"
    "fmt"
    "github.com/kierdavis/mealplanner/mpdb"
    "github.com/kierdavis/mealplanner/mphandlers"
    "github.com/kierdavis/mealplanner/mpresources"
    "log"
    "net/http"
    "os"
    _ "github.com/go-sql-driver/mysql"
)
var (
    dbSource    = flag.String("dbsource", "", "database source, in the form USER:PASS@unix(/PATH/TO/
        SOCKET)/DB or USER:PASS@tcp(HOST:PORT)/DB")
    host        = flag.String("host", "", "hostname to listen on")
    port        = flag.Int("port", 80, "port to listen on")
    debug       = flag.Bool("debug", false, "debug mode")
    testdata    = flag.Bool("testdata", false, "clear the database and insert test data")
    resourceDir = flag.String("resourcedir", "", "path to directory containing the resources used by
         the application")
)
func main() {
    flag.Parse()
    source := *dbSource
    if source == "" {
        source = os.Getenv("MPDBSOURCE")
        if source == "" {
            fmt.Println("Please specify a non-empty -dbsource flag or set the MPDBSOURCE environment
                 variable.")
            os.Exit(1)
        }
    }
    resDir := *resourceDir
    if resDir == "" {
        resDir = os.Getenv("MPRESDIR")
    }
    mpdb.DBSource = source
    mpresources.SetResourceDir(resDir)
    mpresources.GetTemplates() // Check that the templates load correctly
    err := mpdb.InitDB(*debug, *testdata)
    if err != nil {
        log.Printf("Database error during startup: %s\n", err)
        os.Exit(1)
    }
    listenAddr := fmt.Sprintf("%s:%d", *host, *port)
    m := mphandlers.CreateMux()
    app := http.Handler(m)
    if *debug {
        app = mphandlers.LoggingHandler{Handler: app}
        log.Printf("Listening on %s\n", listenAddr)
    }
    err = http.ListenAndServe(listenAddr, app)
    if err != nil {
        log.Printf("Server error in HTTP listener: %s\n", err)
        os.Exit(1)
    }
}
```

F.1.2 Listing of mpapi/deletemeal.go

```
package mpapi
```

```
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdb"
    "log"
    "net/url"
    "strconv"
)
// deleteMeal handles an API call to delete a meal. Expected parameters: mealid.
// Returns: nothing.
func deleteMeal(params url.Values) (response JSONResponse) {
    mealID, err := strconv.ParseUint(params.Get("mealid"), 10, 64)
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'mealid' parameter"}
    }
    err = mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            err = mpdb.DeleteServingsOf(tx, mealID)
            if err != nil {
                return err
            }
            return mpdb.DeleteMealWithTags(tx, mealID)
        })
    })
    if err != nil {
        log.Printf("Database error: %s\n", err.Error())
        return JSONResponse{Error: "Database error"}
    }
    return JSONResponse{Success: nil}
}
```

F.1.3 Listing of mpapi/deleteserving.go

```
package mpapi
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "github.com/kierdavis/mealplanner/mpdb"
    "log"
    "net/url"
    "strconv"
    "time"
)
// deleteServing handles an API call to delete a meal serving. Expected
// parameters: mealplanid, date. Returns: nothing.
func deleteServing(params url.Values) (response JSONResponse) {
    mpID, err := strconv.ParseUint(params.Get("mealplanid"), 10, 64)
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'mealplanid' parameter"}
    }
    servingDate, err := time.Parse(mpdata.JSONDateFormat, params.Get("date"))
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'date' parameter"}
    }
    err = mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            return mpdb.DeleteServing(tx, mpID, servingDate)
        })
    })
    if err != nil {
        log.Printf("Database error: %s\n", err.Error())
        return JSONResponse{Error: "Database error"}
    }
    return JSONResponse{Success: nil}
}
```

F.1.4 Listing of mpapi/fetchalltags.go

```
package mpapi
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdb"
    "log"
    "net/url"
)
```

```
// fetchAllTags handles an API call to obtain a list of all tags present in the
// database, without duplicates and in alphabetical order. Expected parameters:
// none. Returns: an array of tags.
func fetchAllTags(params url.Values) (response JSONResponse) {
    var tags []string
    err := mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            tags, err = mpdb.ListAllTags(tx, true)
            return err
        })
    })
    if err != nil {
        log.Printf("Database error: %s\n", err.Error())
        return JSONResponse{Error: "Database error"}
    }
    return JSONResponse{Success: tags}
}
```

F.1.5 Listing of mpapi/fetchmeallist.go

```
package mpapi
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "github.com/kierdavis/mealplanner/mpdb"
    "log"
    "net/url"
    "regexp"
)
var wordRegexp = regexp.MustCompile("\\w+")
// fetchMealList handles an API call to fetch a list of all meals in the
// database. Expected parameters: none. Returns: an array of meal/tags objects.
func fetchMealList(params url.Values) (response JSONResponse) {
    query := params.Get("query")
    var words []string
    if query != "" {
        words = wordRegexp.FindAllString(query, -1)
    }
    var mts []mpdata.MealWithTags
    err := mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            if query == "" {
                mts, err = mpdb.ListMealsWithTags(tx, true)
            } else {
                mts, err = mpdb.SearchMealsWithTags(tx, words, true)
            }
            return err
        })
    })
    if err != nil {
        log.Printf("Database error: %s\n", err.Error())
        return JSONResponse{Error: "Database error"}
    }
    return JSONResponse{Success: mts}
}
```

F.1.6 Listing of mpapi/fetchmealplans.go

```
package mpapi
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "github.com/kierdavis/mealplanner/mpdb"
    "log"
    "net/url"
    "time"
)
// fetchMealPlans handles an API call to return a list of meal plans that
// overlap with a specified inclusive date range. Expected parameters: from, to.
// Returns: an array of meal plan objects.
func fetchMealPlans(params url.Values) (response JSONResponse) {
    from, err := time.Parse(mpdata.JSONDateFormat, params.Get("from"))
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'from' parameter"}
```

```
    }
    to, err := time.Parse(mpdata.JSONDateFormat, params.Get("to"))
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'to' parameter"}
    }
    var mps []*mpdata.MealPlan
    err = mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            mps, err = mpdb.ListMealPlansBetween(tx, from, to)
            return err
        })
    })
    if err != nil {
        log.Printf("Database error: %s\n", err.Error())
        return JSONResponse{Error: "Database error"}
    }
    return JSONResponse{Success: mps}
}
```

F.1.7 Listing of mpapi/fetchservings.go

```
package mpapi
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "github.com/kierdavis/mealplanner/mpdb"
    "log"
    "net/url"
    "strconv"
)
// fetchServingsRecord is a structure to hold the result objects returned
// by the fetch servings API call in order for them to be encoded as JSON.
type fetchServingsRecord struct {
    Date     string `json:"date"`
    HasMeal  bool   `json:"hasmeal"`
    MealID   uint64 `json:"mealid"`
    MealName string `json:"mealname"`
}
// fetchServings handles an API call to list all the servings for a given meal
// plan. Expected parameters: mealplanid. Returns: an array of
// fetchServingsRecord objects.
func fetchServings(params url.Values) (response JSONResponse) {
    mpID, err := strconv.ParseUint(params.Get("mealplanid"), 10, 64)
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'mealplanid' parameter"}
    }
    var results []*fetchServingsRecord
    err = mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            mps, err := mpdb.GetMealPlanWithServings(tx, mpID)
            if err != nil {
                return err
            }
            if mps.MealPlan == nil {
                return nil
            }
            for _, date := range mps.MealPlan.Days() {
                ts := &fetchServingsRecord{
                    Date: date.Format(mpdata.JSONDateFormat),
                }
                for _, serving := range mps.Servings {
                    if serving.Date == date {
                        ts.HasMeal = true
                        ts.MealID = serving.MealID
                        meal, err := mpdb.GetMeal(tx, serving.MealID)
                        if err != nil {
                            return err
                        }
                        if meal == nil {
                            log.Printf("Warning: meal plan %d -> serving %s points to nonexistent
                                meal %d\n", mpID, date.Format("2006-01-02"), serving.MealID)
                            ts.MealName = "???"
                        } else {
                            ts.MealName = meal.Name
                        }
```

```
                        break
                    }
                }
                results = append(results, ts)
            }
            return err
        })
    })
    if err != nil {
        log.Printf("Database error: %s\n", err.Error())
        return JSONResponse{Error: "Database error"}
    }
    return JSONResponse{Success: results}
}
```

F.1.8 Listing of mpapi/fetchsuggestions.go

```
package mpapi
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "github.com/kierdavis/mealplanner/mpdb"
    "log"
    "net/url"
    "strconv"
    "time"
)
// fetchSuggestions handles an API call to generate suggestions for a given date.
// Expected parameters: date. Returns: an array of suggestion objects.
func fetchSuggestions(params url.Values) (response JSONResponse) {
    mpID, err := strconv.ParseUint(params.Get("mealplanid"), 10, 64)
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'mealplanid' parameter"}
    }
    dateServed, err := time.Parse(mpdata.JSONDateFormat, params.Get("date"))
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'date' parameter"}
    }
    var suggs []*mpdata.Suggestion
    err = mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            suggs, err = mpdb.GenerateSuggestions(tx, mpID, dateServed)
            return err
        })
    })
    if err != nil {
        log.Printf("Database error: %s\n", err.Error())
        return JSONResponse{Error: "Database error"}
    }
    return JSONResponse{Success: suggs}
}
```

F.1.9 Listing of mpapi/mpapi.go

```
package mpapi
import (
    "encoding/json"
    "log"
    "net/http"
    "net/url"
)
// JSONResponse contains the response structure returned to the client.
// If the 'Error' field is nonempty, the response indicates an error has
// occurred, else the response is assumed to be a successful one.
type JSONResponse struct {
    Error   string      `json:"error"`   // The error message, in the event of an unsuccessful
        response.
    Success interface{} `json:"success"` // The response payload, in the event of a successful
        response.
}
// HandleAPICall handles an HTTP request for an API call. It obtains the form
// values, passes them through Dispatch and sends the resulting JSON response
// to the client.
func HandleAPICall(w http.ResponseWriter, r *http.Request) {
```

```
    var response JSONResponse
    err := r.ParseForm()
    if err != nil {
        response = JSONResponse{Error: "Could not parse request body."}
    } else {
        response = Dispatch(r.Form)
    }
    w.Header().Set("Content-Type", "application/json")
    err = json.NewEncoder(w).Encode(response)
    if err != nil {
        log.Printf("Error: Could not write JSON response: %s\n", err.Error())
    }
}
// Dispatch inspects the "command" parameter and dispatches the request to the
// appropriate handler function.
func Dispatch(params url.Values) (response JSONResponse) {
    switch params.Get("command") {
    case "fetch-meal-list":
        return fetchMealList(params)
    case "toggle-favourite":
        return toggleFavourite(params)
    case "delete-meal":
        return deleteMeal(params)
    case "fetch-all-tags":
        return fetchAllTags(params)
    case "fetch-servings":
        return fetchServings(params)
    case "fetch-suggestions":
        return fetchSuggestions(params)
    case "update-serving":
        return updateServing(params)
    case "delete-serving":
        return deleteServing(params)
    case "update-notes":
        return updateNotes(params)
    case "fetch-meal-plans":
        return fetchMealPlans(params)
    }
    return JSONResponse{Error: "Invalid or missing command"}
}
```

F.1.10 Listing of mpapi/togglefavourite.go

```
package mpapi
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdb"
    "log"
    "net/url"
    "strconv"
)
// toggleFavourite implements an API call to toggle the "favourite" status of
// a given meal. Expected paramaters: mealid. Returns: the updated "favourite"
// status of the meal.
func toggleFavourite(params url.Values) (response JSONResponse) {
    mealID, err := strconv.ParseUint(params.Get("mealid"), 10, 64)
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'mealid' parameter"}
    }
    var isFavourite bool
    err = mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            isFavourite, err = mpdb.ToggleFavourite(tx, mealID)
            return err
        })
    })
    if err != nil {
        log.Printf("Database error: %s\n", err.Error())
        return JSONResponse{Error: "Database error"}
    }
    return JSONResponse{Success: isFavourite}
}
```

F.1.11 Listing of mpapi/updatenotes.go

```
package mpapi
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdb"
    "log"
    "net/url"
    "strconv"
)
// updateNotes implements an API call to update the notes associated with a
// meal plan. Expected parameters: mealplanid, notes. Returns: nothing.
func updateNotes(params url.Values) (response JSONResponse) {
    mpID, err := strconv.ParseUint(params.Get("mealplanid"), 10, 64)
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'mealplanid' parameter"}
    }
    notes := params.Get("notes")
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'notes' parameter"}
    }
    err = mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            return mpdb.UpdateNotes(tx, mpID, notes)
        })
    })
    if err != nil {
        log.Printf("Database error: %s\n", err.Error())
        return JSONResponse{Error: "Database error"}
    }
    return JSONResponse{Success: nil}
}
```

F.1.12 Listing of mpapi/updateserving.go

```
package mpapi
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "github.com/kierdavis/mealplanner/mpdb"
    "log"
    "net/url"
    "strconv"
    "time"
)
// updateServing implements an API call to update a meal serving for a meal
// plan with a new meal ID, removing the old serving if it already exists.
// Expected parameters: mealplanid, date, mealid. Returns: nothing.
func updateServing(params url.Values) (response JSONResponse) {
    mpID, err := strconv.ParseUint(params.Get("mealplanid"), 10, 64)
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'mealplanid' parameter"}
    }
    dateServed, err := time.Parse(mpdata.JSONDateFormat, params.Get("date"))
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'date' parameter"}
    }
    mealID, err := strconv.ParseUint(params.Get("mealid"), 10, 64)
    if err != nil {
        return JSONResponse{Error: "Invalid or missing 'mealid' parameter"}
    }
    err = mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            err = mpdb.DeleteServing(tx, mpID, dateServed)
            if err != nil {
                return err
            }
            s := &mpdata.Serving{
                MealPlanID: mpID,
                Date:       dateServed,
                MealID:     mealID,
            }
            return mpdb.AddServing(tx, s)
        })
    })
```

```
    if err != nil {
        log.Printf("Database error: %s\n", err.Error())
        return JSONResponse{Error: "Database error"}
    }
    return JSONResponse{Success: nil}
}
```

F.1.13 Listing of mpdata/mealplanjson.go

```
package mpdata
import (
    "encoding/json"
    "time"
)
// MarshalJSON encodes a meal plan into its JSON form.
func (mp *MealPlan) MarshalJSON() (text []byte, err error) {
    mpj := mealPlanJSON{
        ID:        mp.ID,
        Notes:     mp.Notes,
        StartDate: mp.StartDate.Format(JSONDateFormat),
        EndDate:   mp.EndDate.Format(JSONDateFormat),
    }
    return json.Marshal(mpj)
}
// UnmarshalJSON populates the fields of the receiver with values parsed from
// the input JSON.
func (mp *MealPlan) UnmarshalJSON(text []byte) (err error) {
    var mpj mealPlanJSON
    err = json.Unmarshal(text, &mpj)
    if err != nil {
        return err
    }
    mp.ID = mpj.ID
    mp.Notes = mpj.Notes
    mp.StartDate, err = time.Parse(JSONDateFormat, mpj.StartDate)
    if err != nil {
        return err
    }
    mp.EndDate, err = time.Parse(JSONDateFormat, mpj.EndDate)
    if err != nil {
        return err
    }
    return nil
}
// MarshalJSON encodes a meal serving into its JSON form.
func (s *Serving) MarshalJSON() (text []byte, err error) {
    sj := servingJSON{
        MealPlanID: s.MealPlanID,
        Date:       s.Date.Format(JSONDateFormat),
        MealID:     s.MealID,
    }
    return json.Marshal(sj)
}
// UnmarshalJSON populates the fields of the receiver with values parsed from
// the input JSON.
func (s *Serving) UnmarshalJSON(text []byte) (err error) {
    var sj servingJSON
    err = json.Unmarshal(text, &sj)
    if err != nil {
        return err
    }
    s.MealPlanID = sj.MealPlanID
    s.MealID = sj.MealID
    s.Date, err = time.Parse(JSONDateFormat, sj.Date)
    if err != nil {
        return err
    }
    return nil
}
```

F.1.14 Listing of mpdata/mpdata.go

```
// Package mpdata defines the data model and core algorithms used by the
// application.
```

```
package mpdata
// DatepickerDateFormat is the time format used by the JQuery datepicker widget
// and sent in POST requests. See also: documentation on 'time.Parse'.
const DatepickerDateFormat = "02/01/2006"
// JSONDateFormat is the time format used in JSON encodings. See also:
// documentation on 'time.Parse'.
const JSONDateFormat = "2006-01-02"
```

F.1.15 Listing of mpdata/score.go

```
package mpdata
import (
    "math"
)
// Scorer encapsulates the scoring algorithm used for suggestion
// generation.
type Scorer struct {
    tagScores map[string]float32
}
// NewScorer allocates and returns a new Scorer.
func NewScorer() (s *Scorer) {
    return &Scorer{
        tagScores: make(map[string]float32),
    }
}
// AddTagScore should be called for every tag occurrence in the database. The
// 'dist' argument refers to the number of days between the date that
// suggestions are being generated for and the date of the closest serving of
// the meal the tag is associated with.
func (s *Scorer) AddTagScore(tag string, dist int) {
    score, ok := s.tagScores[tag]
    if !ok {
        score = 1.0 // the default
    }
    score *= 0.1 + float32(math.Tanh(float64(dist)*0.2))
    s.tagScores[tag] = score
}
// ScoreSuggestion calculates a score for the given suggestion and assigns it
// to the 'Score' field of the argument.
func (s *Scorer) ScoreSuggestion(sugg *Suggestion) {
    score := float32(1)
    if sugg.CSD < 0 {
        score *= 1.6
    } else {
        score *= 1.45 - (2.8 / float32(sugg.CSD+1))
    }
    for _, tag := range sugg.MT.Tags {
        tagScore, ok := s.tagScores[tag]
        if ok {
            score *= tagScore
        }
    }
    if sugg.MT.Meal.Favourite {
        score *= 2.0
    }
    sugg.Score = score
}
```

F.1.16 Listing of mpdata/suggestionslice.go

```
package mpdata
// SuggestionSlice is an implementation of 'sort.Interface', allowing a
// list of suggestions to be sorted by their score.
type SuggestionSlice []*Suggestion
// Len returns the number of suggestions in the list.
func (ss SuggestionSlice) Len() (n int) {
    return len(ss)
}
// Less returns whether the suggestion indexed by 'i' has a higher score than
// the suggestion indexed by 'j' and so should be placed closer to the start of
// the list.
func (ss SuggestionSlice) Less(i int, j int) (less bool) {
    return ss[i].Score > ss[j].Score
}
```

```
// Swap exchanges the suggestions indexed by 'i' and 'j'.
func (ss SuggestionSlice) Swap(i int, j int) {
    ss[i], ss[j] = ss[j], ss[i]
}
```

F.1.17 Listing of mpdata/types.go

```
package mpdata
import (
    "time"
)
// Meal holds information about a meal in the database.
type Meal struct {
    ID         uint64 `json:"id"`         // The database's unique identifier for the meal.
    Name       string `json:"name"`       // The name of the meal.
    RecipeURL string `json:"recipe"`     // The possibly empty URL of the recipe for the meal.
    Favourite bool   `json:"favourite"` // Whether or not the meal is marked as a favourite.
}
// MealWithTags pairs a Meal with its associated tags.
type MealWithTags struct {
    Meal *Meal     `json:"meal"` // The meal.
    Tags []string `json:"tags"` // The meal's tags.
}
// Suggestion pairs a Meal with its associated tags, closest serving distance and score.
type Suggestion struct {
    MT    MealWithTags `json:"mt"`    // The meal and tags.
    CSD   int          `json:"-"`     // The closest serving distance (used in computing the score).
    Score float32      `json:"score"` // The meal's score.
}
// MealPlan holds information about a meal plan in the database. It
// contains no JSON field tags as the mealPlanJSON struct is actually used for
// encoding/decoding; however, the MarshalJSON/UnmarshalJSON methods take care
// of this.
type MealPlan struct {
    ID        uint64    // The database's unique identifier for the meal plan.
    Notes     string    // The textual notes associated with the meal plan.
    StartDate time.Time // The date of the first day in the meal plan.
    EndDate   time.Time // The date of the last day in the meal plan.
}
// mealPlanJSON is the intermediate struct used for JSON encoding/decoding
// of a meal plan. An intermediate type is used as the time.Times need to be
// encoded in a specific format.
type mealPlanJSON struct {
    ID        uint64 `json:"id"`
    Notes     string `json:"notes"`
    StartDate string `json:"startdate"`
    EndDate   string `json:"enddate"`
}
// Days returns a slice of times representing the days between mp.StartDate and
// mp.EndDate, inclusive.
func (mp *MealPlan) Days() (days []time.Time) {
    curr := mp.StartDate
    for !curr.After(mp.EndDate) {
        days = append(days, curr)
        curr = curr.Add(time.Hour * 24)
    }
    return days
}
// Serving holds information about a serving of a meal in the database.
// It contains no JSON field tags as the servingJSON struct is actually used for
// encoding/decoding; however, the MarshalJSON/UnmarshalJSON methods take care
// of this.
type Serving struct {
    MealPlanID uint64
    Date       time.Time
    MealID     uint64
}
// servingJSON is the intermediate struct used for JSON encoding/decoding
// of a meal plan. An intermediate type is used as the time.Time needs to be
// encoded in a specific format.
type servingJSON struct {
    MealPlanID uint64 `json:"mealplanid"`
    Date       string `json:"date"`
    MealID     uint64 `json:"mealid"`
}
```

```
// MealPlanWithServings pairs a MealPlan with its associated Servings.
type MealPlanWithServings struct {
    MealPlan *MealPlan  `json:"mealplan"`
    Servings []*Serving `json:"servings"`
}
```

F.1.18 Listing of mpdb/meal.go

```
package mpdb
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
)
const SearchTextFunc = "CONCAT(meal.name, ' ', meal.recipe, ' ', IFNULL((SELECT GROUP_CONCAT(tag.tag
     SEPARATOR ' ') FROM tag WHERE tag.mealid = meal.id), ''))"
// SQL statement for listing meals.
const ListMealsSQL = "SELECT meal.id, meal.name, meal.recipe, meal.favourite FROM meal"
// SQL statement for listing meals sorted by name.
const ListMealsByNameSQL = ListMealsSQL + " ORDER BY meal.name"
const CreateSearchPatternsTableSQL = "CREATE TEMPORARY TABLE search_patterns (pattern VARCHAR(255))"
const InsertSearchPatternSQL = "INSERT INTO search_patterns VALUES (?)"
const DropSearchPatternsTableSQL = "DROP TABLE search_patterns"
// SQL statement for fetching information about a meal.
const GetMealSQL = "SELECT meal.name, meal.recipe, meal.favourite FROM meal WHERE meal.id = ?"
// SQL statement for fetching tags associated with a meal.
const GetMealTagsSQL = "SELECT tag.tag FROM tag WHERE tag.mealid = ?"
const UpdateSearchTextSQL = "UPDATE meal SET meal.searchtext = " + SearchTextFunc + " WHERE meal.id
    = ?"
// SQL statement for adding a meal.
const AddMealSQL = "INSERT INTO meal VALUES (NULL, ?, ?, ?, '')"
// SQL statement for updating the information about a meal.
const UpdateMealSQL = "UPDATE meal SET meal.name = ?, meal.recipe = ?, meal.favourite = ? WHERE meal
    .id = ?"
// SQL statement for deleting all tags associated with a meal.
const DeleteMealTagsSQL = "DELETE FROM tag WHERE tag.mealid = ?"
// SQL statement for adding a tag to a meal.
const AddMealTagSQL = "INSERT INTO tag VALUES (?, ?)"
// SQL statement for testing whether a meal is marked as a favourite.
const IsFavouriteSQL = "SELECT meal.favourite FROM meal WHERE meal.id = ?"
// SQL statement to set the "favourite" status of a meal.
const SetFavouriteSQL = "UPDATE meal SET meal.favourite = ? WHERE meal.id = ?"
// SQL statement to delete a meal.
const DeleteMealSQL = "DELETE FROM meal WHERE meal.id = ?"
// SQL statement to list all tags in the database.
const ListAllTagsSQL = "SELECT DISTINCT tag.tag FROM tag"
// SQL statement to list all tags in the database sorted by name.
const ListAllTagsByNameSQL = "SELECT DISTINCT tag.tag FROM tag ORDER BY tag.tag ASC"
// ListMeals fetches and returns a list of all meals in the database. If the
// parameter 'sortByName' is true, the meals are sorted in alphabetical order
// by name.
func ListMeals(q Queryable, sortByName bool) (meals []*mpdata.Meal, err error) {
    var query string
    if sortByName {
        query = ListMealsByNameSQL
    } else {
        query = ListMealsSQL
    }
    rows, err := q.Query(query)
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    return readMeals(rows)
}
// ListMealsWithTags fetches and returns a list of all meals in the database
// with their associated tags. If the parameter 'sortByName' is true, the meals
// are sorted in alphabetical order by name.
func ListMealsWithTags(q Queryable, sortByName bool) (mts []mpdata.MealWithTags, err error) {
    meals, err := ListMeals(q, sortByName)
    if err != nil {
        return nil, err
    }
    return AttachMealTags(q, meals)
}
func SearchMeals(q Queryable, words []string, sortByName bool) (meals []*mpdata.Meal, err error) {
```

```
    query := "SELECT meal.id, meal.name, meal.recipe, meal.favourite FROM meal"
    conjuctive := "WHERE"
    args := make([]interface{}, len(words))
    for i, word := range words {
        query += " " + conjuctive + " meal.searchtext LIKE ?"
        args[i] = "%" + word + "%"
        conjuctive = "AND"
    }
    if sortByName {
        query += " ORDER BY meal.name"
    }
    rows, err := q.Query(query, args...)
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    return readMeals(rows)
}
func SearchMealsWithTags(q Queryable, words []string, sortByName bool) (mts []mpdata.MealWithTags,
    err error) {
    meals, err := SearchMeals(q, words, sortByName)
    if err != nil {
        return nil, err
    }
    return AttachMealTags(q, meals)
}
func readMeals(rows *sql.Rows) (meals []*mpdata.Meal, err error) {
    for rows.Next() {
        meal := &mpdata.Meal{}
        err = rows.Scan(&meal.ID, &meal.Name, &meal.RecipeURL, &meal.Favourite)
        if err != nil {
            return nil, err
        }
        meals = append(meals, meal)
    }
    err = rows.Err()
    if err != nil {
        return nil, err
    }
    return meals, nil
}
func AttachMealTags(q Queryable, meals []*mpdata.Meal) (mts []mpdata.MealWithTags, err error) {
    getTagsStmt, err := q.Prepare(GetMealTagsSQL)
    if err != nil {
        return nil, err
    }
    defer getTagsStmt.Close()
    for _, meal := range meals {
        tags, err := getMealTagsPrepared(getTagsStmt, meal.ID)
        if err != nil {
            return nil, err
        }
        mt := mpdata.MealWithTags{
            Meal: meal,
            Tags: tags,
        }
        mts = append(mts, mt)
    }
    return mts, nil
}
// GetMeal fetches information from the database about the meal identified by
// 'mealID'.
func GetMeal(q Queryable, mealID uint64) (meal *mpdata.Meal, err error) {
    meal = &mpdata.Meal{ID: mealID}
    err = q.QueryRow(GetMealSQL, mealID).Scan(&meal.Name, &meal.RecipeURL, &meal.Favourite)
    if err != nil {
        if err == sql.ErrNoRows {
            return nil, nil
        }
        return nil, err
    }
    return meal, nil
}
// GetMealTags fetches the list of tags associated with the meal identified by
// 'mealID'.
func GetMealTags(q Queryable, mealID uint64) (tags []string, err error) {
```

```
    rows, err := q.Query(GetMealTagsSQL, mealID)
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    return readTags(rows)
}
// getMealTagsPrepared fetches the list of tags associated with the meal
// identified by 'mealID'. 'stmt' is assumed to be a prepared statement compiled
// from GetMealTagsSQL.
func getMealTagsPrepared(stmt *sql.Stmt, mealID uint64) (tags []string, err error) {
    rows, err := stmt.Query(mealID)
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    return readTags(rows)
}
// GetMealWithTags combines GetMeal and GetMealTags.
func GetMealWithTags(q Queryable, mealID uint64) (mt mpdata.MealWithTags, err error) {
    meal, err := GetMeal(q, mealID)
    if err != nil {
        return mpdata.MealWithTags{}, err
    }
    tags, err := GetMealTags(q, mealID)
    if err != nil {
        return mpdata.MealWithTags{}, err
    }
    return mpdata.MealWithTags{Meal: meal, Tags: tags}, nil
}
// AddMeal adds the information in 'meal' to the database as a new record, then
// sets 'meal.ID' to the identifier of this new record.
func AddMeal(q Queryable, meal *mpdata.Meal) (err error) {
    err = addMeal(q, meal)
    if err != nil {
        return err
    }
    return UpdateSearchText(q, meal.ID)
}
func addMeal(q Queryable, meal *mpdata.Meal) (err error) {
    result, err := q.Exec(AddMealSQL, meal.Name, meal.RecipeURL, meal.Favourite)
    if err != nil {
        return err
    }
    mealID, err := result.LastInsertId()
    if err != nil {
        return err
    }
    meal.ID = uint64(mealID)
    return nil
}
func UpdateSearchText(q Queryable, mealID uint64) (err error) {
    _, err = q.Exec(UpdateSearchTextSQL, mealID)
    return err
}
// AddMealTags adds the the list of tags given in 'tags' to the meal identified
// by 'mealID'.
func AddMealTags(q Queryable, mealID uint64, tags []string) (err error) {
    err = addMealTags(q, mealID, tags)
    if err != nil {
        return err
    }
    return UpdateSearchText(q, mealID)
}
func addMealTags(q Queryable, mealID uint64, tags []string) (err error) {
    stmt, err := q.Prepare(AddMealTagSQL)
    if err != nil {
        return err
    }
    defer stmt.Close()
    for _, tag := range tags {
        _, err = stmt.Exec(mealID, tag)
        if err != nil {
            return err
        }
    }
```

```
    return nil
}
// AddMealWithTags combines 'AddMeal' and 'AddMealTags'.
func AddMealWithTags(q Queryable, mt mpdata.MealWithTags) (err error) {
    err = addMealWithTags(q, mt)
    if err != nil {
        return err
    }
    return UpdateSearchText(q, mt.Meal.ID)
}
func addMealWithTags(q Queryable, mt mpdata.MealWithTags) (err error) {
    err = addMeal(q, mt.Meal)
    if err != nil {
        return err
    }
    return addMealTags(q, mt.Meal.ID, mt.Tags)
}
// UpdateMeal replaces with the information in the database for the meal
// identified by 'meal.ID' with the information in 'meal'.
func UpdateMeal(q Queryable, meal *mpdata.Meal) (err error) {
    err = updateMeal(q, meal)
    if err != nil {
        return err
    }
    return UpdateSearchText(q, meal.ID)
}
func updateMeal(q Queryable, meal *mpdata.Meal) (err error) {
    _, err = q.Exec(UpdateMealSQL, meal.Name, meal.RecipeURL, meal.Favourite, meal.ID)
    return err
}
// DeleteMealTags deletes all tags in the database associated with the meal
// identified by 'mealID'. If no such tags exist, no error is raised.
func DeleteMealTags(q Queryable, mealID uint64) (err error) {
    err = deleteMealTags(q, mealID)
    if err != nil {
        return err
    }
    return UpdateSearchText(q, mealID)
}
func deleteMealTags(q Queryable, mealID uint64) (err error) {
    _, err = q.Exec(DeleteMealTagsSQL, mealID)
    return err
}
// UpdateMealTags replaces the tags associated with the meal identified by
// 'mealID' with the list given by 'tags'.
func UpdateMealTags(q Queryable, mealID uint64, tags []string) (err error) {
    err = updateMealTags(q, mealID, tags)
    if err != nil {
        return err
    }
    return UpdateSearchText(q, mealID)
}
func updateMealTags(q Queryable, mealID uint64, tags []string) (err error) {
    err = deleteMealTags(q, mealID)
    if err != nil {
        return err
    }
    return addMealTags(q, mealID, tags)
}
// UpdateMealWithTags combines UpdateMeal and UpdateMealTags.
func UpdateMealWithTags(q Queryable, mt mpdata.MealWithTags) (err error) {
    err = updateMealWithTags(q, mt)
    if err != nil {
        return err
    }
    return UpdateSearchText(q, mt.Meal.ID)
}
func updateMealWithTags(q Queryable, mt mpdata.MealWithTags) (err error) {
    err = updateMeal(q, mt.Meal)
    if err != nil {
        return err
    }
    return updateMealTags(q, mt.Meal.ID, mt.Tags)
}
// ToggleFavourite toggles the "favourite" status of the meal identified by
// 'mealID', and returns the new favourite status.
```

```
func ToggleFavourite(q Queryable, mealID uint64) (isFavourite bool, err error) {
    err = q.QueryRow(IsFavouriteSQL, mealID).Scan(&isFavourite)
    if err != nil {
        return false, err
    }
    isFavourite = !isFavourite
    _, err = q.Exec(SetFavouriteSQL, isFavourite, mealID)
    return isFavourite, err
}
// DeleteMeal deletes the meal record identified by 'mealID'. If no such meal
// exists, no error is raised.
func DeleteMeal(q Queryable, mealID uint64) (err error) {
    _, err = q.Exec(DeleteMealSQL, mealID)
    return err
}
// DeleteMealWithTags deletes the meal record identified by 'mealID', and all
// tag records associated with it.
func DeleteMealWithTags(q Queryable, mealID uint64) (err error) {
    err = DeleteMeal(q, mealID)
    if err != nil {
        return err
    }
    return deleteMealTags(q, mealID)
}
// ListAllTags returns a list (without duplicates) of all tags that appear in
// the database. If the 'sortByName' parameter is true, the tags are sorted into
// alphabetical order.
func ListAllTags(q Queryable, sortByName bool) (tags []string, err error) {
    var query string
    if sortByName {
        query = ListAllTagsByNameSQL
    } else {
        query = ListAllTagsSQL
    }
    rows, err := q.Query(query)
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    return readTags(rows)
}
// readTags reads a *sql.Rows as produced by GetMealTags or
// getMealTagsPrepared and converts it into a slice of tags.
func readTags(rows *sql.Rows) (tags []string, err error) {
    var tag string
    for rows.Next() {
        err = rows.Scan(&tag)
        if err != nil {
            return nil, err
        }
        tags = append(tags, tag)
    }
    err = rows.Err()
    if err != nil {
        return nil, err
    }
    return tags, nil
}
```

F.1.19 Listing of mpdb/mealplan.go

```
package mpdb
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "time"
)
// SQL statement for retrieving information about a meal plan.
const GetMealPlanSQL = "SELECT mealplan.notes, mealplan.startdate, mealplan.enddate FROM mealplan
    WHERE mealplan.id = ?"
// SQL statement for adding a meal plan to the database.
const AddMealPlanSQL = "INSERT INTO mealplan VALUES (NULL, ?, ?, ?)"
// SQL statement for setting the notes associated with a meal plan.
const UpdateNotesSQL = "UPDATE mealplan SET mealplan.notes = ? WHERE mealplan.id = ?"
// SQL statement for deleting a meal plan identified by its ID.
```

```
const DeleteMealPlanSQL = "DELETE FROM mealplan WHERE mealplan.id = ?"
// SQL statement for listing meal plans that overlap a given date range.
const ListMealPlansBetweenSQL = "SELECT mealplan.id, mealplan.startdate, mealplan.enddate " +
    "FROM mealplan " +
    "WHERE ? <= mealplan.enddate && mealplan.startdate <= ?"
// SQL statement for retrieving information about a meal serving.
const GetServingSQL = "SELECT serving.mealid FROM serving WHERE serving.mealplanid = ? AND serving.
    dateserved = ?"
// SQL statement for retrieving the servings associated with a meal plan.
const GetServingsSQL = "SELECT serving.dateserved, serving.mealid FROM serving WHERE serving.
    mealplanid = ?"
// SQL statement for returning the number of servings on a meal plan.
const CountServingsSQL = "SELECT COUNT(serving.dateserved) FROM serving WHERE serving.mealplanid = ?
    "
// SQL statement for deleting a serving.
const DeleteServingSQL = "DELETE FROM serving WHERE serving.mealplanid = ? AND serving.dateserved =
    ?"
// SQL statement for deleting all servings in a meal plan.
const DeleteServingsSQL = "DELETE FROM serving WHERE serving.mealplanid = ?"
// SQL statement for deleting all servings of a given meal.
const DeleteServingsOfSQL = "DELETE FROM serving WHERE serving.mealid = ?"
// SQL statement for adding a meal serving.
const InsertServingSQL = "INSERT INTO serving VALUES (?, ?, ?)"
// GetMealPlan returns information about the meal plan identified by 'mpID'.
func GetMealPlan(q Queryable, mpID uint64) (mp *mpdata.MealPlan, err error) {
    mp = &mpdata.MealPlan{ID: mpID}
    err = q.QueryRow(GetMealPlanSQL, mpID).Scan(&mp.Notes, &mp.StartDate, &mp.EndDate)
    if err != nil {
        if err == sql.ErrNoRows {
            return nil, nil
        }
        return nil, err
    }
    return mp, nil
}
// AddMealPlan adds the information contained in 'mp' to the database as a new
// meal plan record. It assigns the identifier of the newly created record to
// the ID field of the meal plan.
func AddMealPlan(q Queryable, mp *mpdata.MealPlan) (err error) {
    result, err := q.Exec(AddMealPlanSQL, mp.Notes, mp.StartDate, mp.EndDate)
    if err != nil {
        return err
    }
    mpID, err := result.LastInsertId()
    if err != nil {
        return err
    }
    mp.ID = uint64(mpID)
    return nil
}
// UpdateNotes sets the notes associated with the meal plan identified by 'mpID'
// to 'notes'.
func UpdateNotes(q Queryable, mpID uint64, notes string) (err error) {
    _, err = q.Exec(UpdateNotesSQL, notes, mpID)
    return err
}
// DeleteMealPlan deletes the meal plan record identified by 'mpID'. If no such
// meal plan exists, no error is raised.
func DeleteMealPlan(q Queryable, mpID uint64) (err error) {
    _, err = q.Exec(DeleteMealPlanSQL, mpID)
    return err
}
// ListMealPlansBetween returns a list of all meal plans in the database whose
// date range (start date to end date) overlaps with the given date range
// ('from' to 'to').
func ListMealPlansBetween(q Queryable, from time.Time, to time.Time) (mps []*mpdata.MealPlan, err
    error) {
    rows, err := q.Query(ListMealPlansBetweenSQL, from, to)
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    for rows.Next() {
        mp := &mpdata.MealPlan{}
        err = rows.Scan(&mp.ID, &mp.StartDate, &mp.EndDate)
        if err != nil {
```

```
            return nil, err
        }
        mps = append(mps, mp)
    }
    err = rows.Err()
    if err != nil {
        return nil, err
    }
    return mps, nil
}
// GetServing returns information about the meal serving identified by the
// meal plan identifier 'mpID' and the serving date 'date'.
func GetServing(q Queryable, mpID uint64, date time.Time) (serving *mpdata.Serving, err error) {
    serving = &mpdata.Serving{MealPlanID: mpID, Date: date}
    err = q.QueryRow(GetServingSQL, mpID, date).Scan(&serving.MealID)
    if err != nil {
        if err == sql.ErrNoRows {
            return nil, nil
        }
        return nil, err
    }
    return serving, nil
}
// GetServings returns a slice containing the servings that are part of the
// meal plan identified by 'mpID'.
func GetServings(q Queryable, mpID uint64) (servings []*mpdata.Serving, err error) {
    rows, err := q.Query(GetServingsSQL, mpID)
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    for rows.Next() {
        serving := &mpdata.Serving{MealPlanID: mpID}
        err = rows.Scan(&serving.Date, &serving.MealID)
        if err != nil {
            return nil, err
        }
        servings = append(servings, serving)
    }
    err = rows.Err()
    if err != nil {
        return nil, err
    }
    return servings, nil
}
// CountServings returns the number of servings in the meal plan identified by
// 'mpID'.
func CountServings(q Queryable, mpID uint64) (numServings int, err error) {
    err = q.QueryRow(CountServingsSQL, mpID).Scan(&numServings)
    if err != nil {
        return 0, err
    }
    return numServings, nil
}
// GetMealPlanWithServings returns the information about the meal plan
// identified by 'mpID' including its servings.
func GetMealPlanWithServings(q Queryable, mpID uint64) (mps *mpdata.MealPlanWithServings, err error)
 {
    mp, err := GetMealPlan(q, mpID)
    if err != nil {
        return nil, err
    }
    servings, err := GetServings(q, mpID)
    if err != nil {
        return nil, err
    }
    mps = &mpdata.MealPlanWithServings{
        MealPlan: mp,
        Servings: servings,
    }
    return mps, nil
}
// DeleteServing deletes the serving at 'date' on the meal plan identified by
// 'mpID'. If no such serving exists, no error is raised.
func DeleteServing(q Queryable, mpID uint64, date time.Time) (err error) {
    _, err = q.Exec(DeleteServingSQL, mpID, date)
```

```
    return err
}
// DeleteServings deletes all servings on the meal plan identified by 'mpID'. If
// no such servings exist, no error is raised.
func DeleteServings(q Queryable, mpID uint64) (err error) {
    _, err = q.Exec(DeleteServingsSQL, mpID)
    return err
}
// DeleteServingsOf deletes all servings of the meal identified by 'mealID'. IF
// no such servings exist, no error is raised.
func DeleteServingsOf(q Queryable, mealID uint64) (err error) {
    _, err = q.Exec(DeleteServingsOfSQL, mealID)
    return err
}
// AddServing adds the information containing in 'serving' to a new serving
// record in the database.
func AddServing(q Queryable, serving *mpdata.Serving) (err error) {
    _, err = q.Exec(InsertServingSQL, serving.MealPlanID, serving.Date, serving.MealID)
    return err
}
// AutoFillMealPlan assigns servings to every day in 'mp' using the top
// suggestion for each day.
func AutoFillMealPlan(q Queryable, mp *mpdata.MealPlan) (err error) {
    for _, date := range mp.Days() {
        err = AutoFillMealPlanDay(q, mp.ID, date)
        if err != nil {
            return err
        }
    }
    return nil
}
// AutoFillMealPlanDay assigns a serving to day 'date' on the meal plan
// identified by 'mpID' using the top suggestion.
func AutoFillMealPlanDay(q Queryable, mpID uint64, date time.Time) (err error) {
    suggs, err := GenerateSuggestions(q, mpID, date)
    if err != nil {
        return err
    }
    err = DeleteServing(q, mpID, date)
    if err != nil {
        return err
    }
    serving := &mpdata.Serving{
        MealPlanID: mpID,
        Date:       date,
        MealID:     suggs[0].MT.Meal.ID,
    }
    return AddServing(q, serving)
}
```

F.1.20 Listing of mpdb/migration.go

```
package mpdb
import (
    "fmt"
    "log"
)
type MigrationError struct {
    From    uint
    To      uint
    Message string
}
func (e MigrationError) Error() (msg string) {
    return e.Message
}
type Migration struct {
    From  uint
    To    uint
    Stmts []string
}
func (m *Migration) Apply(q Queryable) (err error) {
    for _, stmt := range m.Stmts {
        _, err = q.Exec(stmt)
        if err != nil {
            return err
```

```
        }
    }
    return nil
}
func FindMigration(from uint, maxTo uint) (m *Migration) {
    var best *Migration
    for _, m = range Migrations {
        if m.From == from && m.To <= maxTo && (best == nil || m.To > best.To) {
            best = m
        }
    }
    return best
}
func GetDatabaseVersion(q Queryable) (v uint, err error) {
    err = q.QueryRow("SELECT version FROM version").Scan(&v)
    return v, err
}
func SetDatabaseVersion(q Queryable, v uint) (err error) {
    _, err = q.Exec("UPDATE version SET version = ?", v)
    return err
}
// Migrate the database from the current version to 'targetVersion'.
func Migrate(q Queryable, targetVersion uint, debug bool) (err error) {
    currentVersion, err := GetDatabaseVersion(q)
    if err != nil {
        return err
    }
    if currentVersion > targetVersion {
        return MigrationError{
            From:    currentVersion,
            To:      targetVersion,
            Message: fmt.Sprintf("Cannot migrate to an earlier version of the database (%d) from the
                current version (%d)", targetVersion, currentVersion),
        }
    }
    if debug {
        log.Printf("Migration: Database is at version %d, migration target is %d. Checking for
            available migrations.\n", currentVersion, targetVersion)
    }
    for currentVersion < targetVersion {
        m := FindMigration(currentVersion, targetVersion)
        if m == nil {
            return MigrationError{
                From:    currentVersion,
                To:      targetVersion,
                Message: fmt.Sprintf("No migration defined between versions %d and %d",
                    currentVersion, targetVersion),
            }
        }
        if debug {
            log.Printf("Migration: Executing migration from version %d to %d.\n", m.From, m.To)
        }
        err = m.Apply(q)
        if err != nil {
            return err
        }
        currentVersion = m.To
        err = SetDatabaseVersion(q, currentVersion)
        if err != nil {
            return err
        }
    }
    if debug {
        log.Printf("Migration: Done. Database is now at version %d.\n", currentVersion)
    }
    return nil
}
const LatestVersion = 1
var Migrations = []*Migration{
    // 2014-02-27 - Add 'searchtext' column to 'meal' table.
    &Migration{0, 1, []string{
        "ALTER TABLE meal ADD COLUMN searchtext TEXT NOT NULL",
        "UPDATE meal SET meal.searchtext = " + SearchTextFunc,
    }},
}
```

F.1.21 Listing of mpdb/mpdb.go

```
// Package mpdb provides routines for manipulating the database whilst
// preserving referential integrity as best as possible.
package mpdb
import (
    "database/sql"
    "fmt"
    "github.com/go-sql-driver/mysql"
    "log"
)
// DBDriver is the driver name used when connecting to the database.
const DBDriver = "mysql"
// DBParams are extra parameters required for the database routines to function.
const DBParams = "?parseTime=true"
// DBSource identifies how to connect to the database. It should take the form
// "USER:PASS@unix(/PATH/TO/SOCKET)/DBNAME" or "USER:PASS@tcp(HOST:PORT)/DBNAME".
// By default, it will attempt to connect via the local Unix socket to the
// 'mealplanner' database, with username 'mealplanner' and no password.
var DBSource = "mealplanner@unix(/var/run/mysqld/mysqld.sock)/mealplanner"
// Queryable represents a type that can be queried (either a *sql.DB
// or *sql.Tx). See documentation on 'database/sql#DB' for information on the
// methods in this interface.
type Queryable interface {
    Exec(string, ...interface{}) (sql.Result, error)
    Prepare(string) (*sql.Stmt, error)
    Query(string, ...interface{}) (*sql.Rows, error)
    QueryRow(string, ...interface{}) *sql.Row
}
// LoggingQueryable wraps a Queryable while logging all executions of its
// functions to standard output. It is intended for debugging purposes.
type LoggingQueryable struct {
    Q Queryable
}
// Exec executes a query without returning any rows. The args are for any
// placeholder parameters in the query.
func (lq LoggingQueryable) Exec(query string, args ...interface{}) (result sql.Result, err error) {
    result, err = lq.Q.Exec(query, args...)
    log.Printf("SQL: Exec(%v, %v) -> %v\n", query, args, err)
    return result, err
}
// Prepare creates a prepared statement for later queries or executions.
// Multiple queries or executions may be run concurrently from the returned
// statement.
func (lq LoggingQueryable) Prepare(query string) (stmt *sql.Stmt, err error) {
    stmt, err = lq.Q.Prepare(query)
    log.Printf("SQL: Prepare(%v) -> %v\n", query, err)
    return stmt, err
}
// Query executes a query that returns rows, typically a SELECT. The args are
// for any placeholder parameters in the query.
func (lq LoggingQueryable) Query(query string, args ...interface{}) (rows *sql.Rows, err error) {
    rows, err = lq.Q.Query(query, args...)
    log.Printf("SQL: Query(%v, %v) -> %v\n", query, args, err)
    return rows, err
}
// QueryRow executes a query that is expected to return at most one row.
// QueryRow always return a non-nil value. Errors are deferred until Row's Scan
// method is called.
func (lq LoggingQueryable) QueryRow(query string, args ...interface{}) (row *sql.Row) {
    row = lq.Q.QueryRow(query, args...)
    log.Printf("SQL: QueryRow(%v, %v) -> %v\n", query, args, row)
    return row
}
// Connect creates a new connection to the database using DBDriver and
// DB_SOURCE.
func Connect() (db *sql.DB, err error) {
    return sql.Open(DBDriver, DBSource+DBParams)
}
// FailedCloseError contains information regarding a situation where an error
// occurs when closing a resource in response to an earlier error.
type FailedCloseError struct {
    What          string // A string used in the error message to identify what resource was being
        closed.
    CloseError    error  // The error returned when the resource was closed.
    OriginalError error  // The original error that triggered the closing of the resource.
```

```
}
// Error formats the information contained in 'err' into an error message.
func (err *FailedCloseError) Error() (msg string) {
    return fmt.Sprintf("%s\nAdditionally, when attempting to %s: %s", err.OriginalError.Error(), err
        .What, err.CloseError.Error())
}
// WithConnectionFunc represents a function that can be used with
// WithConnection.
type WithConnectionFunc func(*sql.DB) error
// WithTransactionFunc represents a function that can be used with
// WithTransaction.
type WithTransactionFunc func(*sql.Tx) error
// WithConnection opens a connection to the database, calls 'f' with the
// database as a parameter, then ensures the database is closed even in the
// event of an error. If an error occurs when closing the database, a
// 'FailedCloseError' is returned.
func WithConnection(f WithConnectionFunc) (err error) {
    // Connect to database
    db, err := Connect()
    if err != nil {
        return err
    }
    // Run the passed function
    err = f(db)
    // Close the database
    closeErr := db.Close()
    // If closing the database caused an error, return a FailedCloseError
    if closeErr != nil {
        err = &FailedCloseError{"close connection", closeErr, err}
    }
    return err
}
// WithTransaction begins a transaction on the given database connection, calls
// 'f' with the transaction as a parameter, then ensures the transaction is
// committed if 'f' completes successfully or rolled back in the event of an
// error. If an error occurs when committing or rolling back the transaction, a
// 'FailedCloseError' is returned.
func WithTransaction(db *sql.DB, f WithTransactionFunc) (err error) {
    // Begin transaction
    tx, err := db.Begin()
    if err != nil {
        return err
    }
    // Run the passed function
    err = f(tx)
    var closeErr error
    var what string
    // Commit or rollback the transaction
    if err != nil {
        closeErr = tx.Rollback()
        what = "roll back transaction"
    } else {
        closeErr = tx.Commit()
        what = "commit transaction"
    }
    // If committing/rolling back the transaction caused an error, return a
    // FailedCloseError
    if closeErr != nil {
        err = &FailedCloseError{what, closeErr, err}
    }
    return err
}
func isNonexistentTableError(err error) bool {
    mysqlError, isMysqlError := err.(*mysql.MySQLError)
    return isMysqlError && mysqlError.Number == 1146
}
```

F.1.22 Listing of mpdb/suggs.go

```
package mpdb
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "sort"
    "time"
```

```
)
// SQL statement to obtain a list of all tags and their distances to every
// serving of the meals they are attached to.
const CalculateTagScoresSQL = "SELECT tag.tag, MIN(ABS(DATEDIFF(serving.dateserved, ?))) " +
    "FROM tag " +
    "INNER JOIN serving ON serving.mealid = tag.mealid " +
    "GROUP BY tag.tag"
    //"WHERE NOT (serving.mealplanid = ? AND serving.dateserved = ?)"
// SQL statement to obtain a list of meals along with the distances to their
// closest servings.
const ListSuggestionsSQL = "SELECT meal.id, meal.name, meal.recipe, meal.favourite, MIN(ABS(DATEDIFF
    (serving.dateserved, ?))) " +
    "FROM meal " +
    "LEFT JOIN serving ON meal.id = serving.mealid " +
    //"WHERE NOT (serving.mealplanid = ? AND serving.dateserved = ?) " +
    "GROUP BY meal.id"
// GenerateSuggestions calculates a score for each meal in the database based on
// their suitability for serving on 'date'. These are returned as a list of
// Suggestions.
func GenerateSuggestions(q Queryable, mpID uint64, date time.Time) (suggs []*mpdata.Suggestion, err
    error) {
    s := mpdata.NewScorer()
    err = calculateTagScores(q, s, mpID, date)
    if err != nil {
        return nil, err
    }
    suggs, err = listSuggestions(q, mpID, date)
    if err != nil {
        return nil, err
    }
    err = getTagsForSuggestions(q, suggs)
    if err != nil {
        return nil, err
    }
    for _, sugg := range suggs {
        s.ScoreSuggestion(sugg)
    }
    sort.Sort(mpdata.SuggestionSlice(suggs))
    // Scale the scores to between 0 and 1
    maxScore := suggs[0].Score
    minScore := suggs[len(suggs)-1].Score
    scoreRange := maxScore - minScore
    for _, sugg := range suggs {
        sugg.Score = (sugg.Score - minScore) / scoreRange
    }
    return suggs, nil
}
// calculateTagScores prepares the scorer 's' by adding a score for each usage
// of a tag.
func calculateTagScores(q Queryable, s *mpdata.Scorer, mpID uint64, date time.Time) (err error) {
    rows, err := q.Query(CalculateTagScoresSQL, date)
    if err != nil {
        return err
    }
    defer rows.Close()
    var tag string
    var dist int
    for rows.Next() {
        err = rows.Scan(&tag, &dist)
        if err != nil {
            return err
        }
        s.AddTagScore(tag, dist)
    }
    err = rows.Err()
    if err != nil {
        return err
    }
    return nil
}
// listSuggestions returns a list of meals (without tags) and the distance
// between 'date' and their closest serving to 'date'.
func listSuggestions(q Queryable, mpID uint64, date time.Time) (suggs []*mpdata.Suggestion, err
    error) {
    rows, err := q.Query(ListSuggestionsSQL, date)
    if err != nil {
```

```
        return nil, err
    }
    defer rows.Close()
    for rows.Next() {
        meal := new(mpdata.Meal)
        sugg := new(mpdata.Suggestion)
        sugg.MT.Meal = meal
        var csd sql.NullInt64
        err = rows.Scan(&meal.ID, &meal.Name, &meal.RecipeURL, &meal.Favourite, &csd)
        if err != nil {
            return nil, err
        }
        if csd.Valid && csd.Int64 != 0 {
            sugg.CSD = int(csd.Int64)
        } else {
            sugg.CSD = -1
        }
        suggs = append(suggs, sugg)
    }
    err = rows.Err()
    if err != nil {
        return nil, err
    }
    return suggs, nil
}
// getTagsForSuggestions fills the tags field of each suggestion in 'suggs'.
func getTagsForSuggestions(q Queryable, suggs []*mpdata.Suggestion) (err error) {
    getTagsStmt, err := q.Prepare(GetMealTagsSQL)
    if err != nil {
        return err
    }
    defer getTagsStmt.Close()
    for _, sugg := range suggs {
        sugg.MT.Tags, err = getMealTagsPrepared(getTagsStmt, sugg.MT.Meal.ID)
        if err != nil {
            return err
        }
    }
    return nil
}
```

F.1.23 Listing of mpdb/tables.go

```
package mpdb
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "log"
    "time"
)
// SQL statements to delete tables.
var DeleteTablesSQLs = []string{
    "DROP TABLE IF EXISTS meal",
    "DROP TABLE IF EXISTS tag",
    "DROP TABLE IF EXISTS mealplan",
    "DROP TABLE IF EXISTS serving",
}
// SQL statements to create tables.
var CreateTablesSQLs = []string{
    "CREATE TABLE IF NOT EXISTS meal ( " +
        "id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT, " +
        "name VARCHAR(255) NOT NULL, " +
        "recipe TEXT, " +
        "favourite BOOLEAN NOT NULL, " +
        "searchtext TEXT NOT NULL, " +
        "PRIMARY KEY (id) " +
        ")",
    "CREATE TABLE IF NOT EXISTS tag ( " +
        "mealid BIGINT UNSIGNED NOT NULL, " +
        "tag VARCHAR(64) NOT NULL, " +
        "PRIMARY KEY (mealid, tag) " +
        ")",
    "CREATE TABLE IF NOT EXISTS mealplan ( " +
        "id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT, " +
        "notes TEXT, " +
```

```
        "startdate DATE NOT NULL, " +
        "enddate DATE NOT NULL, " +
        "PRIMARY KEY (id) " +
        ")",
    "CREATE TABLE IF NOT EXISTS serving ( " +
        "mealplanid BIGINT UNSIGNED NOT NULL, " +
        "dateserved DATE NOT NULL, " +
        "mealid BIGINT UNSIGNED NOT NULL, " +
        "PRIMARY KEY (mealplanid, dateserved) " +
        ")",
}
// SQL statements to clear tables.
var ClearTablesSQLs = []string{
    "DELETE FROM meal",
    "DELETE FROM tag",
    "DELETE FROM mealplan",
    "DELETE FROM serving",
}
// execList runs a list of SQL statements, discarding the results.
func execList(q Queryable, queries []string) (err error) {
    for _, query := range queries {
        _, err = q.Exec(query)
        if err != nil {
            return err
        }
    }
    return nil
}
// DeleteTables drops the database tables if they exist.
func DeleteTables(q Queryable) (err error) {
    return execList(q, DeleteTablesSQLs)
}
// CreateTables creates the database tables if they do not exist.
func CreateTables(q Queryable) (err error) {
    return execList(q, CreateTablesSQLs)
}
func InitialiseVersion(q Queryable, debug bool) (err error) {
    var version uint
    err = q.QueryRow("SELECT version FROM version").Scan(&version)
    isNTE := isNonexistentTableError(err)
    if err == nil { // All is fine.
        if debug {
            log.Printf("Version check: OK, current version is %d\n", version)
        }
        return nil
    } else if isNTE || err == sql.ErrNoRows { // No version set.
        if debug {
            log.Printf("Version check: version not set yet\n")
        }
        if isNTE { // 'version' table does not exist.
            if debug {
                log.Printf("Version check: creating version table\n")
            }
            _, err = q.Exec("CREATE TABLE version (version INT UNSIGNED NOT NULL)")
            if err != nil {
                return err
            }
        }
        // Check if other tables exist.
        _, err = q.Exec("SELECT meal.id FROM meal LIMIT 1")
        if err == nil { // Table 'meal' exists.
            if debug {
                log.Printf("Version check: assuming first startup since introduction of versioning\n
                    ")
            }
            version = 0
        } else if isNonexistentTableError(err) { // Table 'meal' does not exist.
            if debug {
                log.Printf("Version check: assuming empty database\n")
            }
            version = LatestVersion
        } else { // Unknown error.
            return err
        }
    } else { // Unknown error.
        return err
```

```
    }
    if debug {
        log.Printf("Version check: setting version to %d\n", version)
    }
    _, err = q.Exec("INSERT INTO version VALUES (?)", version)
    return err
}
// ClearTables deletes all records from the entire database.
func ClearTables(q Queryable) (err error) {
    return execList(q, ClearTablesSQLs)
}
// InitDB creates the database tables if they don't exist. If 'debug' is true,
// debug messages are printed. If 'testData' is true, the tables are also
// cleared and test data are added to them.
func InitDB(debug bool, testData bool) (err error) {
    return WithConnection(func(db *sql.DB) (err error) {
        return WithTransaction(db, func(tx *sql.Tx) (err error) {
            err = InitialiseVersion(tx, debug)
            if err != nil {
                return err
            }
            err = CreateTables(tx)
            if err != nil {
                return err
            }
            err = Migrate(tx, LatestVersion, debug)
            if err != nil {
                return err
            }
            if testData {
                if debug {
                    log.Printf("Clearing database and inserting test data.\n")
                }
                err = ClearTables(tx)
                if err != nil {
                    return err
                }
                err = InsertTestData(tx)
                if err != nil {
                    return err
                }
            }
            return nil
        })
    })
}
// InsertTestData inserts some predefined meals and meal plans into the
// database for testing purposes.
func InsertTestData(q Queryable) (err error) {
    err = AddMealWithTags(q, mpdata.MealWithTags{
        Meal: &mpdata.Meal{
            Name:      "Chilli con carne",
            RecipeURL: "http://example.net/chilli",
            Favourite: false,
        },
        Tags: []string{
            "spicy",
            "lentil",
            "rice",
        },
    })
    if err != nil {
        return err
    }
    err = AddMealWithTags(q, mpdata.MealWithTags{
        Meal: &mpdata.Meal{
            Name:      "Carrot and lentil soup",
            RecipeURL: "http://example.net/soup",
            Favourite: false,
        },
        Tags: []string{
            "lentil",
            "soup",
            "quick",
        },
    })
```

```
    if err != nil {
        return err
    }
    err = AddMealWithTags(q, mpdata.MealWithTags{
        Meal: &mpdata.Meal{
            Name:      "Nachos",
            RecipeURL: "http://example.net/nachos",
            Favourite: true,
        },
        Tags: []string{
            "spicy",
            "mexican",
        },
    })
    if err != nil {
        return err
    }
    mp1 := &mpdata.MealPlan{
        Notes:     "some notes",
        StartDate: time.Date(2014, time.January, 25, 0, 0, 0, 0, time.UTC),
        EndDate:   time.Date(2014, time.February, 4, 0, 0, 0, 0, time.UTC),
    }
    err = AddMealPlan(q, mp1)
    if err != nil {
        return err
    }
    mp2 := &mpdata.MealPlan{
        Notes:     "some other notes",
        StartDate: time.Date(2014, time.February, 5, 0, 0, 0, 0, time.UTC),
        EndDate:   time.Date(2014, time.February, 8, 0, 0, 0, 0, time.UTC),
    }
    err = AddMealPlan(q, mp2)
    if err != nil {
        return err
    }
    log.Printf("Test meal plans are %d and %d\n", mp1.ID, mp2.ID)
    return nil
}
```

F.1.24 Listing of mphandlers/addmeal.go

```
package mphandlers
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "github.com/kierdavis/mealplanner/mpdb"
    "net/http"
    "strconv"
)
// handleAddMealForm handles HTTP requests for the "new meal" form.
func handleAddMealForm(w http.ResponseWriter, r *http.Request) {
    renderTemplate(w, "edit-meal-form.html", nil)
}
// handleAddMealAction handles HTTP requests for submission of the "new meal"
// form.
func handleAddMealAction(w http.ResponseWriter, r *http.Request) {
    // Parse the POST request body
    err := r.ParseForm()
    if err != nil {
        serverError(w, err)
        return
    }
    // Create a MealWithTags value from the form fields
    mt := mpdata.MealWithTags{
        Meal: &mpdata.Meal{
            Name:      r.FormValue("name"),
            RecipeURL: r.FormValue("recipe"),
            Favourite: r.FormValue("favourite") != "",
        },
        Tags: r.Form["tags"],
    }
    // Add the record to the database
    err = mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            return mpdb.AddMealWithTags(tx, mt)
```

```
        })
    })
    if err != nil {
        serverError(w, err)
        return
    }
    // Redirect to list of meals
    redirect(w, http.StatusSeeOther, "/meals?highlight="+strconv.FormatUint(mt.Meal.ID, 10))
}
```

F.1.25 Listing of mphandlers/browsemealplans.go

```
package mphandlers
import (
    "net/http"
    "time"
)
// showingData contains the information passed to the meal plan browser
// template regarding which year/month to display.
type showingData struct {
    Year  int
    Month int
}
// handleBrowseMealPlans handles HTTP requests for the meal plan browser.
func handleBrowseMealPlans(w http.ResponseWriter, r *http.Request) {
    showing := time.Now()
    showingStr := r.FormValue("showing")
    if showingStr != "" {
        var err error
        showing, err = time.Parse("2006-01-02", showingStr)
        if err != nil {
            showing = time.Now()
        }
    }
    sd := showingData{showing.Year(), int(showing.Month() - 1)}
    renderTemplate(w, "browse-mps.html", sd)
}
```

F.1.26 Listing of mphandlers/browsemeals.go

```
package mphandlers
import (
    //"database/sql"
    //"github.com/kierdavis/mealplanner/mpdata"
    //"github.com/kierdavis/mealplanner/mpdb"
    "net/http"
    "strconv"
)
// highlightData contains information passed to the meal browser template
// regarding which meal, if any, should be highlighted in the list.
type highlightData struct {
    Highlight bool
    MealID    uint64
}
// handleBrowseMeals handles HTTP requests for the meal list.
func handleBrowseMeals(w http.ResponseWriter, r *http.Request) {
    var hd highlightData
    highlightStr := r.FormValue("highlight")
    if highlightStr != "" {
        mealID, err := strconv.ParseUint(highlightStr, 10, 64)
        if err != nil {
            httpError(w, BadRequestError)
            return
        }
        hd.Highlight = true
        hd.MealID = mealID
    }
    renderTemplate(w, "browse-meals.html", hd)
}
```

F.1.27 Listing of mphandlers/createmealplan.go

```go
package mphandlers
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "github.com/kierdavis/mealplanner/mpdb"
    "net/http"
    "strconv"
    "time"
)
// handleCreateMealPlanForm handles HTTP requests for the meal plan creation
// form.
func handleCreateMealPlanForm(w http.ResponseWriter, r *http.Request) {
    renderTemplate(w, "create-mp-form.html", nil)
}
// handleCreateMealPlanAction handles HTTP requests for the submission of the
// meal plan creation form.
func handleCreateMealPlanAction(w http.ResponseWriter, r *http.Request) {
    // Parse the POST request body
    err := r.ParseForm()
    if err != nil {
        serverError(w, err)
        return
    }
    startDate, err := time.Parse(mpdata.DatepickerDateFormat, r.FormValue("start"))
    if err != nil {
        httpError(w, BadRequestError)
        return
    }
    endDate, err := time.Parse(mpdata.DatepickerDateFormat, r.FormValue("end"))
    if err != nil {
        httpError(w, BadRequestError)
        return
    }
    if startDate.After(endDate) {
        httpError(w, BadRequestError)
        return
    }
    auto := r.FormValue("auto") == "true"
    // Create a MealPlan object
    mp := &mpdata.MealPlan{
        StartDate: startDate,
        EndDate:   endDate,
    }
    err = mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            // Add mp to the database
            err = mpdb.AddMealPlan(tx, mp)
            if err != nil {
                return err
            }
            // Optionally fill the meal plan automatically
            if auto {
                err = mpdb.AutoFillMealPlan(tx, mp)
                if err != nil {
                    return err
                }
            }
            return nil
        })
    })
    if err != nil {
        serverError(w, err)
        return
    }
    redirect(w, http.StatusSeeOther, "/mealplans/"+strconv.FormatUint(mp.ID, 10)+"/edit")
}
```

F.1.28 Listing of mphandlers/deletemealplan.go

```go
package mphandlers
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "github.com/kierdavis/mealplanner/mpdb"
    "net/http"
```

```
)
// deleteMPData contains information passed to the meal plan deletion template
// regarding the meal plan that is being deleted.
type deleteMPData struct {
    MP          *mpdata.MealPlan
    NumServings int
}
// handleDeleteMealPlanForm handles HTTP requests for the meal plan deletion
// confirmation page.
func handleDeleteMealPlanForm(w http.ResponseWriter, r *http.Request) {
    mpID, ok := getUint64Var(r, "mealplanid")
    if !ok {
        httpError(w, BadRequestError)
        return
    }
    var mp *mpdata.MealPlan
    var numServings int
    err := mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            mp, err = mpdb.GetMealPlan(tx, mpID)
            if err != nil {
                return err
            }
            numServings, err = mpdb.CountServings(tx, mpID)
            return err
        })
    })
    if err != nil {
        serverError(w, err)
        return
    }
    if mp == nil {
        httpError(w, NotFoundError)
        return
    }
    renderTemplate(w, "delete-mp-form.html", deleteMPData{mp, numServings})
}
// handleDeleteMealPlanAction handles HTTP requests for submission of the
// meal plan deletion form.
func handleDeleteMealPlanAction(w http.ResponseWriter, r *http.Request) {
    mpID, ok := getUint64Var(r, "mealplanid")
    if !ok {
        httpError(w, BadRequestError)
        return
    }
    err := mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            err = mpdb.DeleteServings(tx, mpID)
            if err != nil {
                return err
            }
            return mpdb.DeleteMealPlan(tx, mpID)
        })
    })
    if err != nil {
        serverError(w, err)
        return
    }
    redirect(w, http.StatusSeeOther, "/mealplans")
}
```

F.1.29 Listing of mphandlers/editmeal.go

```
package mphandlers
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "github.com/kierdavis/mealplanner/mpdb"
    "net/http"
    "strconv"
)
// handleEditMealForm handles HTTP requests for the "edit meal" form.
func handleEditMealForm(w http.ResponseWriter, r *http.Request) {
    mealID, ok := getUint64Var(r, "mealid")
    if !ok {
```

```
        httpError(w, BadRequestError)
        return
    }
    var mt mpdata.MealWithTags
    err := mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            mt, err = mpdb.GetMealWithTags(tx, mealID)
            return err
        })
    })
    if err != nil {
        serverError(w, err)
        return
    }
    if mt.Meal == nil {
        httpError(w, NotFoundError)
        return
    }
    renderTemplate(w, "edit-meal-form.html", mt)
}
// handleEditMealAction handles HTTP requests for submission of the "edit meal"
// form.
func handleEditMealAction(w http.ResponseWriter, r *http.Request) {
    // Get the meal ID from the URL
    mealID, ok := getUint64Var(r, "mealid")
    if !ok {
        httpError(w, BadRequestError)
        return
    }
    // Parse the POST request body
    err := r.ParseForm()
    if err != nil {
        serverError(w, err)
        return
    }
    // Create a MealWithTags value from the form fields
    mt := mpdata.MealWithTags{
        Meal: &mpdata.Meal{
            ID:        mealID,
            Name:      r.FormValue("name"),
            RecipeURL: r.FormValue("recipe"),
            Favourite: r.FormValue("favourite") != "",
        },
        Tags: r.Form["tags"],
    }
    // Update the database record
    err = mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            return mpdb.UpdateMealWithTags(tx, mt)
        })
    })
    if err != nil {
        serverError(w, err)
        return
    }
    // Redirect to list of meals
    redirect(w, http.StatusSeeOther, "/meals?highlight="+strconv.FormatUint(mealID, 10))
}
```

F.1.30 Listing of mphandlers/editmealplan.go

```
package mphandlers
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "github.com/kierdavis/mealplanner/mpdb"
    "net/http"
)
// handleEditMealPlan handles HTTP requests for the meal plan editor.
func handleEditMealPlan(w http.ResponseWriter, r *http.Request) {
    mpID, ok := getUint64Var(r, "mealplanid")
    if !ok {
        httpError(w, BadRequestError)
        return
    }
```

```
    var mp *mpdata.MealPlan
    err := mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            mp, err = mpdb.GetMealPlan(tx, mpID)
            return err
        })
    })
    if err != nil {
        serverError(w, err)
        return
    }
    if mp == nil {
        httpError(w, NotFoundError)
        return
    }
    renderTemplate(w, "edit-mp-form.html", mp)
}
```

F.1.31 Listing of mphandlers/home.go

```
package mphandlers
import (
    "net/http"
)
// handleHome handles HTTP requests for the homepage.
func handleHome(w http.ResponseWriter, r *http.Request) {
    renderTemplate(w, "home.html", nil)
}
```

F.1.32 Listing of mphandlers/httperror.go

```
package mphandlers
import (
    "net/http"
)
// HTTPError holds related information about an HTTP status code used by the
// application.
type HTTPError struct {
    Status    int    // The HTTP status code.
    ShortDesc string // The associated "reason" message sent with the status code.
    LongDesc  string // A longer message displayed to the user on the HTML error page.
}
// BadRequestError represents an HTTP 400 Bad Request error.
var BadRequestError = &HTTPError{
    Status:    http.StatusBadRequest,
    ShortDesc: "Bad Request",
    LongDesc:  "We're sorry, there was an error when processing your request.",
}
// NotFoundError represents an HTTP 404 Not Found error.
var NotFoundError = &HTTPError{
    Status:    http.StatusNotFound,
    ShortDesc: "Not Found",
    LongDesc:  "We're sorry, the page you were looking for was not found on ther server.",
}
// InternalServerError represents an HTTP 500 Internal Server Error.
var InternalServerError = &HTTPError{
    Status:    http.StatusInternalServerError,
    ShortDesc: "Internal Server Error",
    LongDesc:  "We're sorry, the server encountered an unexpected error and was unable to complete
        the request.",
}
```

F.1.33 Listing of mphandlers/logging.go

```
package mphandlers
import (
    "log"
    "net/http"
)
// LoggingHandler wraps an http.Handler, printing a message to standard output
// whenever a request is handled.
type LoggingHandler struct {
```

```
    Handler http.Handler
}
// ServeHTTP handles an HTTP request.
func (lh LoggingHandler) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    lw := &loggingWriter{writer: w, status: 200}
    lh.Handler.ServeHTTP(lw, r)
    lh.logRequest(lw, r)
}
// logRequest logs a request to standard output.
func (lh LoggingHandler) logRequest(lw *loggingWriter, r *http.Request) {
    log.Printf("%s %s -> %d %s from %s\n", r.Method, r.URL.String(), lw.status, http.StatusText(lw.
        status), r.RemoteAddr)
}
// loggingWriter wraps an http.ResponseWriter, recording the status code sent
// to the client and the total number of bytes written.
type loggingWriter struct {
    writer       http.ResponseWriter
    status       int
    totalWritten int
}
// Header returns the header map that will be sent by WriteHeader.
func (lw *loggingWriter) Header() (h http.Header) {
    return lw.writer.Header()
}
// Write writes the data to the connection as part of an HTTP reply.
func (lw *loggingWriter) Write(buffer []byte) (n int, err error) {
    n, err = lw.writer.Write(buffer)
    lw.totalWritten += n
    return n, err
}
// WriteHeader sends an HTTP response header with status code.
func (lw *loggingWriter) WriteHeader(status int) {
    lw.status = status
    lw.writer.WriteHeader(status)
}
```

F.1.34 Listing of mphandlers/mphandlers.go

```
// Package mphandlers defines the HTTP handlers for the application.
package mphandlers
import (
    "github.com/gorilla/mux"
    "github.com/kierdavis/mealplanner/mpapi"
    "github.com/kierdavis/mealplanner/mpresources"
    "net/http"
)
// CreateMux creates a *mux.Router and attaches the application's HTTP handlers
// to it.
func CreateMux() (m *mux.Router) {
    m = mux.NewRouter()
    // Static files
    staticHandler := http.StripPrefix("/static/", http.FileServer(http.Dir(mpresources.GetStaticDir
        ())))
    m.PathPrefix("/static/").Handler(staticHandler)
    // Dynamic handlers
    m.Path("/").Methods("GET", "HEAD").HandlerFunc(handleHome)
    m.Path("/meals").Methods("GET", "HEAD").HandlerFunc(handleBrowseMeals)
    m.Path("/mealplans").Methods("GET", "HEAD").HandlerFunc(handleBrowseMealPlans)
    m.Path("/mealplans/{mealplanid:[0-9]+}").Methods("GET", "HEAD").HandlerFunc(handleViewMealPlan)
    m.Path("/mealplans/{mealplanid:[0-9]+}/edit").Methods("GET", "HEAD").HandlerFunc(
        handleEditMealPlan)
    m.Path("/api").Methods("POST").HandlerFunc(mpapi.HandleAPICall)
    addMeal := m.Path("/meals/new").Subrouter()
    addMeal.Methods("GET", "HEAD").HandlerFunc(handleAddMealForm)
    addMeal.Methods("POST").HandlerFunc(handleAddMealAction)
    editMeal := m.Path("/meals/{mealid:[0-9]+}/edit").Subrouter()
    editMeal.Methods("GET", "HEAD").HandlerFunc(handleEditMealForm)
    editMeal.Methods("POST").HandlerFunc(handleEditMealAction)
    createMP := m.Path("/mealplans/new").Subrouter()
    createMP.Methods("GET", "HEAD").HandlerFunc(handleCreateMealPlanForm)
    createMP.Methods("POST").HandlerFunc(handleCreateMealPlanAction)
    deleteMP := m.Path("/mealplans/{mealplanid:[0-9]+}/delete").Subrouter()
    deleteMP.Methods("GET", "HEAD").HandlerFunc(handleDeleteMealPlanForm)
    deleteMP.Methods("POST").HandlerFunc(handleDeleteMealPlanAction)
    return m
```

```
}
```

F.1.35 Listing of mphandlers/util.go

```
package mphandlers
import (
    "fmt"
    "github.com/gorilla/mux"
    "github.com/kierdavis/mealplanner/mpresources"
    "log"
    "net/http"
    "runtime"
    "strconv"
)
// httpError sends an HTTP error code to the client followed by an HTML error
// page explaining the error.
func httpError(w http.ResponseWriter, h *HTTPError) {
    w.Header().Set("Content-Type", "text/html; charset=utf8")
    w.WriteHeader(h.Status)
    err := mpresources.GetTemplates().ExecuteTemplate(w, "error.html", h)
    if err != nil {
        log.Printf("Internal error when rendering error.html: %s\n", err.Error())
    }
}
// serverError logs 'err' to the console, then sends a 500 Internal Server Error
// response to the client.
func serverError(w http.ResponseWriter, err error) {
    log.Printf("Internal error: %s\n", err.Error())
    _, filename, lineno, ok := runtime.Caller(1)
    if ok {
        log.Printf("  at %s line %d\n", filename, lineno)
    }
    httpError(w, InternalServerError)
}
// redirect sends a redirection response to the client with the given status
// code.
func redirect(w http.ResponseWriter, status int, url string) {
    w.Header().Set("Content-Type", "text/plain; charset=utf8")
    w.Header().Set("Location", url)
    w.WriteHeader(status)
    fmt.Fprintf(w, "Redirecting to %s...\n", url)
}
// renderTemplate renders the named template and returns the rendered HTML to
// the client.
func renderTemplate(w http.ResponseWriter, name string, data interface{}) {
    w.Header().Set("Content-Type", "text/html; charset=utf8")
    err := mpresources.GetTemplates().ExecuteTemplate(w, name, data)
    if err != nil {
        serverError(w, err)
    }
}
// getUint64Var gets a URL variable (a parameter embedded in the request URI)
// and parses it as an unsigned 64-bit integer.
func getUint64Var(r *http.Request, name string) (value uint64, ok bool) {
    vars := mux.Vars(r)
    str, ok := vars[name]
    if !ok {
        return 0, false
    }
    value, err := strconv.ParseUint(str, 10, 64)
    if err != nil {
        return 0, false
    }
    return value, true
}
```

F.1.36 Listing of mphandlers/viewmealplan.go

```
package mphandlers
import (
    "database/sql"
    "github.com/kierdavis/mealplanner/mpdata"
    "github.com/kierdavis/mealplanner/mpdb"
    "net/http"
```

```
)
// handleViewMealPlan handles HTTP requests for the meal plan viewer.
func handleViewMealPlan(w http.ResponseWriter, r *http.Request) {
    mpID, ok := getUint64Var(r, "mealplanid")
    if !ok {
        httpError(w, BadRequestError)
        return
    }
    var mp *mpdata.MealPlan
    err := mpdb.WithConnection(func(db *sql.DB) (err error) {
        return mpdb.WithTransaction(db, func(tx *sql.Tx) (err error) {
            mp, err = mpdb.GetMealPlan(tx, mpID)
            return err
        })
    })
    if err != nil {
        serverError(w, err)
        return
    }
    if mp == nil {
        httpError(w, NotFoundError)
        return
    }
    renderTemplate(w, "view-mp.html", mp)
}
```

F.1.37 Listing of mpresources/mpresources.go

```
// Package mpresources contains the HTML templates and static files used by the
// application.
package mpresources
import (
    "go/build"
    "html/template"
    "path/filepath"
)
// resourceDir is the directory that all resources are stored in. It should be
// considered uninitialised until GetResourceDir is called or it is otherwise
// assigned to.
var resourceDir string
// getSourceDir gets the directory that the source files for this package are
// installed to.
func getSourceDir() (dir string) {
    pkginfo, err := build.Import("github.com/kierdavis/mealplanner/mpresources", "", build.FindOnly)
    if err != nil {
        panic("Resource directory not set and no suitable directory found in the GOPATH")
    }
    return pkginfo.Dir
}
// GetResourceDir returns the resource directory. If it is uninitialised, it
// looks for the package's source directory in the GOPATH and uses that.
func GetResourceDir() (dir string) {
    if resourceDir == "" {
        resourceDir = filepath.Join(getSourceDir(), "resources")
    }
    return resourceDir
}
// SetResourceDir sets the resource directory.
func SetResourceDir(dir string) {
    resourceDir = dir
}
// GetStaticDir returns the directory used for storing static files.
func GetStaticDir() (dir string) {
    return filepath.Join(GetResourceDir(), "static")
}
var templates *template.Template
// GetTemplates loads the templates from the resource directory if they have not
// been loaded already, and returns them.
func GetTemplates() (t *template.Template) {
    if templates == nil {
        pattern := filepath.Join(GetResourceDir(), "templates", "*")
        templates = template.Must(template.ParseGlob(pattern))
    }
    return templates
}
```

```
/*
// ResourcesDir is the directory that all resources are stored in.
var ResourcesDir = filepath.Join(getSourceDir(), "resources")
// TemplatesDir is the directory that the templates are stored in.
var TemplatesDir = filepath.Join(ResourcesDir, "templates")
// StaticDir is the directory that static files are stored in.
var StaticDir = filepath.Join(ResourcesDir, "static")
// Templates contains the parsed templates. See also: documentation on
// 'html/template'.
var Templates = template.Must(template.ParseGlob(filepath.Join(TemplatesDir, "*")))
*/
```

F.2 HTML templates

F.2.1 Listing of `browse-meals.html`

```
{{/*
    browse-meals.html contains the list of meals page.
    Dot is expected to be a struct/map containing a field 'Highlight' of type
    bool and a field 'MealID' of integral type.
*/}}
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Browse Meals :: Meal Planner</title>
        {{template "common-head.inc.html"}}
        <script type="text/javascript" src="/static/js/meallistview.js"></script>
        <script type="text/javascript">
            $(document).ready(function() {
                $("#add-meal").click(function() {
                    location.href = "/meals/new";
                });
                var view = new MealListView($("#results"));
                view.addColumn(new MealListViewColumns.NameColumn("meal-list-name"));
                view.addColumn(new MealListViewColumns.TagsColumn("meal-list-tags"));
                view.addColumn(new MealListViewColumns.ActionsColumn("meal-list-actions"));
                var highlighted = false;
                view.setFetchDataCallback(function(params, cb) {
                    MealResult.fetchMealList(params, function(items) {
                        cb(items);
                        if (!highlighted) {
                            {{if .Highlight}}
                                view.highlightItemByID({{.MealID}});
                            {{end}}
                            highlighted = true;
                        }
                    });
                });
                view.render();
            });
        </script>
    </head>
    <body>
        <div class="container">
            <div class="page-header">
                <div class="row">
                    <div class="col-md-8">
                        <ul class="navigation">
                            <li class="home"><a href="/">Home</a></li>
                            <li>Meals</li>
                        </ul>
                    </div>
                </div>
                <div class="row">
                    <div class="col-md-8">
                        <h1>Browse Meals</h1>
                    </div>
                    <div class="col-md-4">
                        <button title="Add a new meal" class="header-button" id="add-meal">
                            <img src="/static/img/add-meal_32x32.png" height="24" alt=""/>
                            Add new meal
                        </button>
                    </div>
```

```
            </div>
        </div>
        <div class="row">
            <div class="col-md-12" id="results">
                <!--
                <img src="/static/img/loading.gif" alt=""/>
                Loading results...
            -->
            </div>
        </div>
        {{template "footer.inc.html"}}
    </div>
</body>
</html>
```

F.2.2 Listing of `browse-mps.html`

```
{{/*
    browse-meal-plans.html contains the meal plan browser page.
    Dot is expected to be of type *time.Time (the month of this time is used
    as the initial month displayed on the calendar).
*/}}
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Browse Meal Plans :: Meal Planner</title>
        {{template "common-head.inc.html"}}
        <script type="text/javascript">
            var cellmap = {};
            function removeTime(date) {
                date.setHours(0);
                date.setMinutes(0);
                date.setSeconds(0);
                date.setMilliseconds(0);
            }
            function renderRows(first, keypoint, last) {
                cellmap = {};
                var container = $("#results").empty();
                var curr = new Date(first.getTime());
                var i, tr, td;
                var today = new Date();
                removeTime(today);
                while (curr <= last) {
                    tr = $("<tr>").appendTo(container);
                    for (i = 0; i < 7; i++) {
                        td = $("<td><div>" + curr.getDate() + "</div></td>").addClass("mpnone").
                            appendTo(tr);
                        if (curr.getMonth() != keypoint.getMonth()) {
                            td.addClass("outside-key-month");
                        }
                        if (curr.getYear() == today.getYear() && curr.getMonth() == today.getMonth()
                             && curr.getDate() == today.getDate()) {
                            td.addClass("today");
                        }
                        cellmap[curr.getTime()] = td;
                        curr.setDate(curr.getDate() + 1);
                    }
                }
            }
            function setCell(date, cls, id) {
                var cell = cellmap[date.getTime()];
                if (MPUtil.nonNull(cell)) {
                    cell.removeClass("mpnone").addClass(cls).addClass("mp-" + id);
                    cell.click(function(event) {
                        event.preventDefault();
                        location.href = "/mealplans/" + id;
                    });
                    cell.mouseover(function() {
                        $(".mp-" + id).addClass("hover");
                    });
                    cell.mouseout(function() {
                        $(".mp-" + id).removeClass("hover");
                    })
                }
            }
```

```
        function addMealPlan(mp) {
            var start = new Date(Date.parse(mp.startdate));
            var end = new Date(Date.parse(mp.enddate));
            removeTime(start);
            removeTime(end);
            var curr = new Date(start.getTime());
            curr.setDate(curr.getDate() + 1);
            setCell(start, "mpstart", mp.id);
            setCell(end, "mpend", mp.id);
            while (curr < end) {
                setCell(curr, "mpmid", mp.id);
                curr.setDate(curr.getDate() + 1);
            }
        }
        function fetchMealPlans(keypoint) {
            var first = new Date(keypoint.getTime());
            first.setDate(1);
            while (first.getDay() != 0) {
                first.setDate(first.getDate() - 1);
            }
            var last = new Date(keypoint.getTime());
            last.setMonth(last.getMonth() + 1);
            last.setDate(0);
            while (last.getDay() != 6) {
                last.setDate(last.getDate() + 1);
            }
            $("#month").text(MPUtil.formatMonthHumanReadable(keypoint));
            renderRows(first, keypoint, last);
            MPAjax.fetchMealPlans(first, last, function(mps) {
                mps = mps || [];
                var i;
                for (i = 0; i < mps.length; i++) {
                    console.log(mps[i]);
                    addMealPlan(mps[i]);
                }
            });
        }
        $(document).ready(function() {
            $("#add-mp").click(function(event) {
                event.preventDefault();
                location.href = "/mealplans/new";
            });
            var keypoint = new Date({{.Year}}, {{.Month}}, 15, 0, 0, 0, 0);
            fetchMealPlans(keypoint);
            $("#next-month").click(function(event) {
                event.preventDefault();
                keypoint.setMonth(keypoint.getMonth() + 1);
                fetchMealPlans(keypoint);
            });
            $("#prev-month").click(function(event) {
                event.preventDefault();
                keypoint.setMonth(keypoint.getMonth() - 1);
                fetchMealPlans(keypoint);
            });
        });
    </script>
</head>
<body>
    <div class="container">
        <div class="page-header">
            <div class="row">
                <div class="col-md-8">
                    <ul class="navigation">
                        <li class="home"><a href="/">Home</a></li>
                        <li>Meal plans</li>
                    </ul>
                </div>
            </div>
            <div class="row">
                <div class="col-md-8">
                    <h1>Browse Meal Plans</h1>
                </div>
                <div class="col-md-4">
                    <button title="Create a new meal plan" class="header-button" id="add-mp">
                        <img src="/static/img/add-meal-plan_32x32.png" height="24" alt=""/>
                        Create meal plan
```

```
                    </button>
                </div>
            </div>
        </div>
        <div class="row">
            <div class="col-md-4"></div>
            <div class="col-md-4">
                <table class="mplist">
                    <thead>
                        <tr>
                            <th colspan="2" style="text-align: left">
                                <button title="Display the previous month of meal plans" id="
                                    prev-month">
                                    <img src="/static/img/prev_24x24.png" height="24" alt="Prev"
                                        />
                                </button>
                            </th>
                            <th colspan="3" style="text-align: center" id="month"></th>
                            <th colspan="2" style="text-align: right">
                                <button title="Display the next month of meal plans" id="next-
                                    month">
                                    <img src="/static/img/next_24x24.png" height="24" alt="Next"
                                        />
                                </button>
                            </th>
                        </tr>
                        <tr>
                            <th>S</th>
                            <th>M</th>
                            <th>T</th>
                            <th>W</th>
                            <th>T</th>
                            <th>F</th>
                            <th>S</th>
                        </tr>
                    </thead>
                    <tbody id="results"></tbody>
                </table>
            </div>
            <div class="col-md-4"></div>
        </div>
        <!--
        <div id="results">
            <img src="/static/img/loading.gif" alt=""/>
            Loading results...
        </div>
        -->
        {{template "footer.inc.html"}}
    </div>
</body>
</html>
```

F.2.3 Listing of `common-head.inc.html`

```
{{/*
    common-head.html contains tags that will appear in the head of all pages.
*/}}
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link href="/static/css/bootstrap.min.css" rel="stylesheet" type="text/css"/>
<!--<link href="/static/css/bootstrap-theme.min.css" rel="stylesheet" type="text/css"/>-->
<link href="/static/css/jquery-ui/jquery-ui-1.10.4.custom.min.css" rel="stylesheet" type="text/css"/
    >
<link href="/static/css/screen.css" rel="stylesheet" type="text/css"/>
<script type="text/javascript" src="/static/js/jquery-1.11.0.min.js"></script>
<script type="text/javascript" src="/static/js/bootstrap.min.js"></script>
<script type="text/javascript" src="/static/js/jquery-ui-1.10.4.custom.min.js"></script>
<script type="text/javascript" src="/static/js/placeholders.jquery.min.js"></script>
<script type="text/javascript" src="/static/js/mpajax.js"></script>
<script type="text/javascript" src="/static/js/mputil.js"></script>
<script type="text/javascript">
    $(document).ready(function() {
        $(".datepicker").datepicker({
            dateFormat: "dd/mm/yy",
        });
        $("input").keypress(function(event) {
```

```
            if (event.keyCode == 13) {
                event.preventDefault();
                $(this).trigger("enterKey");
            }
        });
    });
</script>
```

F.2.4 Listing of create-mp-form.html

```
{{/*
    create-mp-form.html contains the form used for creating meal plans.
*/}}
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Create Meal Plan :: Meal Planner</title>
        {{template "common-head.inc.html"}}
        <script type="text/javascript">
            function validateStartDate(start) {
                var ok = true, msg = "";
                if (start == null) {
                    ok = false;
                    msg = "Please use the format 'dd/mm/yyyy'.";
                }
                if (ok) {
                    $("#start-validation").slideUp();
                }
                else {
                    $("#start-validation td").text(msg);
                    $("#start-validation").slideDown();
                }
                return ok;
            }
            function validateEndDate(start, end) {
                var ok = true, msg = "";
                if (end == null) {
                    ok = false;
                    msg = "Please use the format 'dd/mm/yyyy'.";
                }
                if (ok && start > end) {
                    ok = false;
                    msg = "The start date cannot be after the end date.";
                }
                if (ok) {
                    $("#end-validation").slideUp();
                }
                else {
                    $("#end-validation td").text(msg);
                    $("#end-validation").slideDown();
                }
                return ok;
            }
            function validateDates() {
                var startStr = $("#start").val();
                var endStr = $("#end").val();
                var start = MPUtil.parseDatepickerDate(startStr);
                var end = MPUtil.parseDatepickerDate(endStr);
                return validateStartDate(start) && validateEndDate(start, end);
            }
            $(document).ready(function() {
                $("#form").submit(function() {
                    if (!validateDates()) return false;
                });
            });
        </script>
    </head>
    <body>
        <div class="container">
            <div class="page-header">
                <div class="row">
                    <div class="col-md-8">
                        <ul class="navigation">
                            <li class="home"><a href="/">Home</a></li>
                            <li><a href="/mealplans">Meal plans</a></li>
```

```
                        <li>Create new</li>
                    </ul>
                </div>
            </div>
            <h1>Create Meal Plan</h1>
        </div>
        <form action="" method="post" id="form">
            <table>
                <tbody>
                    <tr>
                        <td>
                            <label for="start">From:</label>
                        </td>
                        <td>
                            <input type="text" name="start" id="start" size="15" class="
                                datepicker" placeholder="dd/mm/yyyy"/>
                        </td>
                    </tr>
                    <tr id="start-validation">
                        <td class="error" colspan="2"></td>
                    </tr>
                    <tr>
                        <td>
                            <label for="end">To:</label>
                        </td>
                        <td>
                            <input type="text" name="end" id="end" size="15" class="datepicker"
                                placeholder="dd/mm/yyyy"/>
                        </td>
                    </tr>
                    <tr id="end-validation">
                        <td class="error" colspan="2"></td>
                    </tr>
                </tbody>
            </table>
            <p style="padding-top: 10px">
                <button title="Reset the form" id="reset" type="reset">
                    <img src="/static/img/reset_16x16.png" height="16" alt=""/>
                    Reset
                </button>
            </p>
            <p style="padding-top: 10px">
                <button title="Create a meal plan manually" type="submit" name="auto" value="
                    false">
                    <img src="/static/img/save_16x16.png" height="16" alt=""/>
                    Create a meal plan from scratch
                </button>
            </p>
            <p style="padding-top: 10px">
                <button title="Create a meal plan automatically, using the top suggestion for
                    each day" type="submit" name="auto" value="true">
                    <img src="/static/img/save_16x16.png" height="16" alt=""/>
                    Generate a meal plan for me
                </button>
            </p>
        </form>
        {{template "footer.inc.html"}}
    </div>
</body>
</html>
```

F.2.5 Listing of `delete-mp-form.html`

```
{{/*
    delete-mp-form.html contains the form used for confirmation of deletion of
    a meal plan.
    Dot is expected to be a struct/map containing a field 'MP' of type
    *mpdata.MealPlan and a field 'NumServings' of integral type.
*/}}
<!DOCTYPE html>
<html>
    <head>
        <title>Delete Meal Plan :: Meal Planner</title>
        {{template "common-head.inc.html"}}
        <script type="text/javascript">
```

```
            $(document).ready(function() {
                $("#no").click(function(event) {
                    event.preventDefault();
                    location.href = "/mealplans/{{.MP.ID}}";
                });
            });
        </script>
    </head>
    <body>
        <div class="container">
            <div class="page-header">
                <div class="row">
                    <div class="col-md-8">
                        <ul class="navigation">
                            <li class="home"><a href="/">Home</a></li>
                            <li><a href="/mealplans">Meal plans</a></li>
                            <li><a href="/mealplans/{{.MP.ID}}">{{.MP.StartDate.Format "02 Jan"}} -
                                {{.MP.EndDate.Format "02 Jan"}}</a></li>
                            <li>Delete</li>
                        </ul>
                    </div>
                </div>
                <div class="row">
                    <div class="col-md-8">
                        <h1>Delete Meal Plan</h1>
                    </div>
                    <div class="col-md-4">
                        <a href="/mealplans/{{.MP.ID}}">Return to meal plan</a>
                    </div>
                </div>
            </div>
            <p>
                Are you sure you want to delete the meal plan spanning
                {{.MP.StartDate.Format "Mon 02 Jan 2006"}} to
                {{.MP.EndDate.Format "Mon 02 Jan 2006"}}?
            </p>
            {{if .NumServings}}
                <p>
                    The {{.NumServings}} servings associated with the meal plan
                    will also be deleted.
                </p>
            {{end}}
            <p>
                <button id="no">
                    <img src="/static/img/back_16x16.png" height="16" alt=""/>
                    No, take me back.
                </button>
                <form action="" method="post">
                    <button id="yes" type="submit">
                        <img src="/static/img/delete_24x24.png" height="16" alt=""/>
                        Yes, delete the meal plan.
                    </button>
                </form>
            </p>
            {{template "footer.inc.html"}}
        </div>
    </body>
</html>
```

F.2.6 Listing of `edit-meal-form.html`

```
{{/*
    edit-meal-form.html contains the form used for adding and editing meals.
    Dot is expected to be of type mpdata.MealWithTags. If empty, the "add meal"
    form is displayed, else the "edit meal" form is displayed.
*/}}
<!DOCTYPE html>
<html lang="en">
    <head>
        {{if .}}
            <title>Edit Meal :: Meal Planner</title>
        {{else}}
            <title>Create Meal :: Meal Planner</title>
        {{end}}
        {{template "common-head.inc.html"}}
```

```
<script type="text/javascript">
    function addTag(tag) {
        var ok = true;
        $("#tags tr").each(function() {
            var thisTag = $(this).data("tag");
            if (thisTag.toLowerCase() == tag.toLowerCase()) {
                ok = false;
            }
        });
        if (ok) {
            var row = $("<tr>").data("tag", tag).appendTo($("#tags"));
            $("<td>").text(tag).appendTo(row);
            $("<button title='Remove this tag from the list' class='remove-tag'><img src='/
                static/img/delete_24x24.png' height='16' alt='Remove'/></button>")
                .appendTo($("<td>").appendTo(row))
                .click(function(event) {
                    event.preventDefault();
                    row.remove();
                });
        }
    }
    function validateMealName(input) {
        var ok = true, msg = "";
        if (input.length < 1 || input.length > 255) {
            ok = false;
            msg = "Meal name must be between 1 and 255 characters in length (inclusive)";
        }
        if (ok) {
            $("#name-validation").slideUp();
        }
        else {
            $("#name-validation td").text(msg);
            $("#name-validation").slideDown();
        }
        return ok;
    }
    function validateTag(input) {
        var ok = true, msg = "";
        if (input.length < 1 || input.length > 64) {
            ok = false;
            msg = "Tag must be between 1 and 64 characters in length (inclusive)";
        }
        if (ok) {
            $("#tag-validation").slideUp();
        }
        else {
            $("#tag-validation td").text(msg);
            $("#tag-validation").slideDown();
        }
        return ok;
    }
    $(document).ready(function() {
        $("#name-validation").hide();
        $("#tag-validation").hide();
        // Bind event handlers
        // Validate upon submitting form.
        $("#form").submit(function() {
            // Validate inputs
            if (!validateMealName($("#name").val())) return false;
            //if (!validateRecipeLink($("#recipe").val())) return false;
            // Send tags on form submission
            $("#tags tr").each(function() {
                var tag = $(this).data("tag");
                $("<input>")
                    .attr("type", "hidden")
                    .attr("name", "tags")
                    .attr("value", tag)
                    .appendTo($("#form"));
            });
        });
        // Implement adding existing tags.
        $("#add-existing-tag").click(function(event) {
            event.preventDefault();
            addTag($("#existing-tag").val());
        });
        // Implement adding new tags.
```

```
            $("#add-new-tag").click(function(event) {
                event.preventDefault();
                var tag = $("#new-tag").val();
                if (!validateTag(tag)) return false;
                addTag(tag);
                $("#new-tag").val("");
            });
            // Implement resetting the tags list on reset.
            $("#reset").click(function() {
                $("#tags").empty();
                {{range .Tags}}
                    addTag("{{. | js}}");
                {{end}}
            });
            // Implement pressing enter in various text boxes;
            $("#name").bind("enterKey", function() {
                $("#form").submit();
            });
            $("#recipe").bind("enterKey", function() {
                $("#form").submit();
            });
            $("#new-tag").bind("enterKey", function() {
                $("#add-new-tag").click();
            });
            // Initialise the tags list.
            $("#reset").click();
            // Initialise existing tags list.
            MPAjax.fetchAllTags(function(tags) {
                MPUtil.renderExistingTagsList(tags, $("#existing-tag"));
            });
        });
    </script>
</head>
<body>
    <div class="container">
        <div class="page-header">
            {{if .}}
                <div class="row">
                    <div class="col-md-8">
                        <ul class="navigation">
                            <li class="home"><a href="/">Home</a></li>
                            <li><a href="/meals">Meals</a></li>
                            <li><a href="/meals?highlight={{.Meal.ID}}">{{.Meal.Name}}</a></li>
                            <li>Edit</li>
                        </ul>
                    </div>
                </div>
                <h1>Edit meal: {{.Meal.Name}}</h1>
            {{else}}
                <div class="row">
                    <div class="col-md-8">
                        <ul class="navigation">
                            <li class="home"><a href="/">Home</a></li>
                            <li><a href="/meals">Meals</a></li>
                            <li>Create new</li>
                        </ul>
                    </div>
                </div>
                <h1>Create meal</h1>
            {{end}}
        </div>
        <form action="" method="post" id="form">
            <div style="margin-bottom: 40px">
                <table>
                    <tr>
                        <td>
                            <label for="name">Name of meal:</label>
                        </td>
                        <td>
                            <input type="text" name="name" id="name" size="50" value="{{.Meal.
                                Name}}"/>
                        </td>
                    </tr>
                    <tr id="name-validation">
                        <td class="error" colspan="2"></td>
                    </tr>
```

```
            <tr>
                <td>
                    <label for="recipe">Link to recipe (optional):</label>
                </td>
                <td>
                    <input type="text" name="recipe" id="recipe" size="50" value="{{.
                        Meal.RecipeURL}}"/>
                </td>
            </tr>
            <tr>
                <td>
                    <label for="favourite">Favourite:</label>
                </td>
                <td>
                    <input type="checkbox" name="favourite" id="favourite" value="yes"
                        {{if .Meal.Favourite}}checked="checked"{{end}} />
                </td>
            </tr>
        </table>
    </div>
    <div style="margin-bottom: 40px">
        <fieldset>
            <legend>Tags</legend>
            <div class="row">
                <div class="col-md-4">
                    <table class="tags-list">
                        <tbody id="tags"></tbody>
                    </table>
                </div>
                <div class="col-md-8">
                    <table>
                        <tr>
                            <td>
                                Add an existing tag:
                            </td>
                            <td>
                                <select id="existing-tag" style="min-width: 100px"></
                                    select>
                            </td>
                            <td>
                                <button title="Add the tag in the drop-down to the list"
                                     id="add-existing-tag">
                                    <img src="/static/img/add_16x16.png" height="16" alt
                                        =""/>
                                    Add
                                </button>
                            </td>
                        </tr>
                        <tr>
                            <td>
                                or add a new tag:
                            </td>
                            <td>
                                <input type="text" id="new-tag" size="15"/>
                            </td>
                            <td>
                                <button title="Add the tag in the text box to the list"
                                    id="add-new-tag">
                                    <img src="/static/img/add_16x16.png" height="16" alt
                                        =""/>
                                    Add
                                </button>
                            </td>
                        </tr>
                        <tr id="tag-validation">
                            <td class="error" colspan="3"></td>
                        </tr>
                    </table>
                </div>
            </div>
        </fieldset>
    </div>
    <p>
        <button title="Reset the form" id="reset" type="reset">
            <img src="/static/img/reset_16x16.png" height="16" alt=""/>
            Reset
```

```
                    </button>
                    <button title="Save the meal to the database" type="submit">
                        <img src="/static/img/save_16x16.png" height="16" alt=""/>
                        Save
                    </button>
                </p>
            </form>
            {{template "footer.inc.html"}}
        </div>
    </body>
</html>
```

F.2.7 Listing of `edit-mp-form.html`

```
{{/*
    edit-mp-form.html contains the form used for editing meal plans.
    Dot is expected to be of type *mpdata.MealPlan.
*/}}
<!DOCTYPE html>
<html>
    <head>
        <title>Edit Meal Plan :: Meal Planner</title>
        {{template "common-head.inc.html"}}
        <script type="text/javascript" src="/static/js/meallistview.js"></script>
        <script type="text/javascript">
            var selectedServingDate = null;
            var selectedServingLink = null;
            function updateServing(item) {
                MPAjax.updateServing({{.ID}}, selectedServingDate, item.id, function() {
                    selectedServingLink.text(item.name);
                });
                $("#dialog").dialog("close");
            }
            function addServingRow(view, result) {
                var date = new Date(Date.parse(result.date));
                var row = $("<tr>");
                $("<td>")
                    .text(MPUtil.formatDateHumanReadable(date))
                    .appendTo(row);
                var a = $("<a href='#'>")
                    .text(result.hasmeal ? result.mealname : "(click to add meal)")
                    .appendTo($("<td>").appendTo(row))
                    .click(function(event) {
                        event.preventDefault();
                        selectedServingDate = date;
                        selectedServingLink = a;
                        view.setFetchDataCallback(function(params, cb) {
                            MealResult.fetchSuggestions({{.ID}}, date, function(items) {
                                cb(items);
                                $("#dialog").dialog("open");
                            });
                        });
                        view.fetchData();
                        /*
                        MPAjax.fetchSuggestions(date, function(suggs) {
                            MPUtil.renderSuggestions(suggs, $("#meal-list"), function(mt) {
                                MPAjax.updateServing({{.ID}}, date, mt.meal.id, function() {
                                    a.text(mt.meal.name);
                                });
                                $("#dialog").dialog("close");
                            });
                            console.log(suggs);
                            $("#dialog").dialog("open");
                        });
                        */
                    });
                $("<button title='Delete this serving'><img src='/static/img/delete_24x24.png'
                    height='16' alt='Delete'/></button>")
                    .appendTo($("<td>").appendTo(row))
                    .click(function(event) {
                        event.preventDefault();
                        MPAjax.deleteServing({{.ID}}, date, function() {
                            a.text("(click to add meal)");
                        });
                    });
```

```
            return row;
        }
        function loadServings(deletedItem) {
            var view = this;
            MPAjax.fetchServings({{.ID}}, function(results) {
                results = results || [];
                var container = $("#servings").empty();
                var i;
                for (i = 0; i < results.length; i++) {
                    container.append(addServingRow(view, results[i]));
                }
            });
        }
        $(document).ready(function() {
            $("#dialog").dialog({
                autoOpen: false,
                draggable: true,
                height: 600,
                modal: true,
                title: "Edit serving",
                width: 400,
            });
            $("#save-notes").click(function(event) {
                event.preventDefault();
                $("#save-text").html("<img src='/static/img/loading.gif' height='16' alt=''/>
                    Saving...");
                $("#save-text").show();
                MPAjax.updateNotes({{.ID}}, $("#notes").val(), function() {
                    $("#save-text").text("Saved!");
                    window.setTimeout(function() {
                        $("#save-text").toggle("fade");
                    }, 3000);
                });
            });
            var view = new MealListView($("#results"));
            view.addColumn(new MealListViewColumns.NameColumn("sugg-list-name"));
            view.addColumn(new MealListViewColumns.TagsColumn("sugg-list-tags"));
            view.addColumn(new MealListViewColumns.ScoreColumn("sugg-list-score"));
            view.addColumn(new MealListViewColumns.ActionsColumn("sugg-list-actions"));
            view.setItemCallback(updateServing);
            view.setDeleteCallback(loadServings);
            view.render();
            loadServings.call(view, null);
        });
    </script>
</head>
<body>
    <div class="container">
        <div class="page-header">
            <div class="row">
                <div class="col-md-8">
                    <ul class="navigation">
                        <li class="home"><a href="/">Home</a></li>
                        <li><a href="/mealplans">Meal plans</a></li>
                        <li><a href="/mealplans/{{.ID}}">{{.StartDate.Format "02 Jan"}} - {{.
                            EndDate.Format "02 Jan"}}</a></li>
                        <li>Edit</li>
                    </ul>
                </div>
            </div>
            <div class="row">
                <div class="col-md-8">
                    <h1>Edit Meal Plan</h1>
                    <h3>{{.StartDate.Format "02 Jan"}} - {{.EndDate.Format "02 Jan"}}</h3>
                </div>
                <div class="col-md-4">
                    <a href="/mealplans/{{.ID}}">Return to meal plan</a>
                </div>
            </div>
        </div>
        <table style="margin-left: 20px; margin-bottom: 10px">
            <thead>
                <tr>
                    <th>Date</th>
                    <th colspan="2">Serving</th>
                </tr>
```

```
                </thead>
                <tbody id="servings"></tbody>
            </table>
            <p style="font-style: italic">
                All changes to the above are saved automatically.
            </p>
            <p>
                <button title="Return to the list of meal plans" id="return">
                    <img src="/static/img/back_16x16.png" height="16" alt=""/>
                    Return to calendar
                </button>
            </p>
            <hr/>
            <div class="row">
                <div class="col-md-8">
                    <textarea id="notes" style="width: 100%">{{.Notes}}</textarea>
                </div>
                <div class="col-md-4">
                    <p>
                        <button title="Save these contents of the text box as the meal plan's notes"
                                id="save-notes">
                            <img src="/static/img/save_16x16.png" height="16" alt=""/>
                            Save notes
                        </button>
                    </p>
                    <p id="save-text"></p>
                </div>
            </div>
            {{template "footer.inc.html"}}
        </div>
        <div id="dialog">
            <div id="results"></div>
        </div>
    </body>
</html>
```

F.2.8 Listing of `error.html`

```
{{/*
    error.html contains a page to display in the event of an HTTP error.
    Dot is expected to be of type *mphandlers.HTTPError.
*/}}
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>{{.ShortDesc}} :: Meal Planner</title>
        {{template "common-head.inc.html"}}
    </head>
    <body>
        <div class="container">
            <div class="page-header">
                <div class="row">
                    <div class="col-md-8">
                        <ul class="navigation">
                            <li class="home"><a href="/">Home</a></li>
                            <li>Error</li>
                        </ul>
                    </div>
                </div>
                <h1>{{.ShortDesc}}</h1>
            </div>
            <p>
                {{.LongDesc}}
            </p>
            <p>
                Please use your browser's Back button to go back and try again.
            </p>
            {{template "footer.inc.html"}}
        </div>
    </body>
</html>
```

F.2.9 Listing of `footer.inc.html`

```
<hr/>
<div class="footer">
    <p>
        UI based on <a href="http://getbootstrap.com/">Bootstrap</a>.
        Graphics from the <a href="http://openiconlibrary.sourceforge.net/">Open Icon Library</a>.
        <a href="https://github.com/kierdavis/mealplanner">Code</a> licensed under
        <a href="https://github.com/kierdavis/mealplanner/blob/master/LICENSE">BSD3</a>.
    </p>
</div>
```

F.2.10 Listing of `home.html`

```
{{/*
    home.html contains the homepage.
*/}}
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Home :: Meal Planner</title>
        {{template "common-head.inc.html"}}
        <script type="text/javascript">
            $(document).ready(function() {
                // Bind event handlers
                $("#create-meal-plan").click(function(event) {
                    event.preventDefault();
                    location.href = "/mealplans/new";
                });
                $("#browse-meal-plans").click(function(event) {
                    event.preventDefault();
                    location.href = "/mealplans";
                });
                $("#add-meal").click(function(event) {
                    event.preventDefault();
                    location.href = "/meals/new";
                });
                $("#browse-meals").click(function(event) {
                    event.preventDefault();
                    location.href = "/meals";
                });
            });
        </script>
    </head>
    <body>
        <div class="container">
            <div class="page-header">
                <div class="row">
                    <div class="col-md-8">
                        <ul class="navigation">
                            <li class="home">Home</li>
                        </ul>
                    </div>
                </div>
                <h1>Home</h1>
            </div>
            <div class="row">
                <div class="col-md-6 homepage-cell">
                    <button class="homepage-button" id="create-meal-plan">
                        <img src="/static/img/add-meal-plan_32x32.png" height="32" alt=""/>
                        Create a meal plan
                    </button>
                </div>
                <div class="col-md-6 homepage-cell">
                    <button class="homepage-button" id="browse-meal-plans">
                        <img src="/static/img/browse-meal-plans_32x32.png" height="32" alt=""/>
                        Browse meal plans
                    </button>
                </div>
            </div>
            <div class="row">
                <div class="col-md-6 homepage-cell">
                    <button class="homepage-button" id="add-meal">
                        <img src="/static/img/add-meal_32x32.png" height="32" alt=""/>
                        Add a meal
                    </button>
                </div>
```

```
                <div class="col-md-6 homepage-cell">
                    <button class="homepage-button" id="browse-meals">
                        <img src="/static/img/browse-meals_32x32.png" height="32" alt=""/>
                        Browse meals
                    </button>
                </div>
            </div>
            {{template "footer.inc.html"}}
        </div>
    </body>
</html>
```

F.2.11 Listing of `view-mp.html`

```
{{/*
    edit-mp-form.html contains the form used for editing meal plans.
    Dot is expected to be of type *mpdata.MealPlan.
*/}}
<!DOCTYPE html>
<html>
    <head>
        <title>Edit Meal Plan :: Meal Planner</title>
        {{template "common-head.inc.html"}}
        <script type="text/javascript">
            function addServingRow(result) {
                var date = new Date(Date.parse(result.date));
                var row = $("<tr>");
                $("<td>")
                    .text(MPUtil.formatDateHumanReadable(date))
                    .appendTo(row);
                $("<td>")
                    .text(result.hasmeal ? result.mealname : "-")
                    .appendTo(row);
                return row;
            }
            $(document).ready(function() {
                $("#edit").click(function(event) {
                    event.preventDefault();
                    location.href = "/mealplans/{{.ID}}/edit";
                });
                $("#delete").click(function(event) {
                    event.preventDefault();
                    location.href = "/mealplans/{{.ID}}/delete";
                });
                MPAjax.fetchServings({{.ID}}, function(results) {
                    results = results || [];
                    var container = $("#servings");
                    var i;
                    for (i = 0; i < results.length; i++) {
                        container.append(addServingRow(results[i]));
                    }
                });
            });
        </script>
    </head>
    <body>
        <div class="container">
            <div class="page-header">
                <div class="row">
                    <div class="col-md-8">
                        <ul class="navigation">
                            <li class="home"><a href="/">Home</a></li>
                            <li><a href="/mealplans?showing={{.StartDate.Format "2006-01-02"}}">Meal
                                plans</a></li>
                            <li>{{.StartDate.Format "02 Jan"}} - {{.EndDate.Format "02 Jan"}}</li>
                        </ul>
                    </div>
                </div>
                <div class="row">
                    <div class="col-md-8">
                        <h1>Meal Plan</h1>
                        <h3>{{.StartDate.Format "02 Jan"}} - {{.EndDate.Format "02 Jan"}}</h3>
                    </div>
                    <div class="col-md-4">
                        <p>
```

```
                        <a href="/mealplans?showing={{.StartDate.Format "2006-01-02"}}">Return
                            to meal plan browser</a>
                    </p>
                    <p>
                        <button title="Edit the meal plan" class="header-button" id="edit">
                            <img src="/static/img/edit_24x24.png" height="24" alt=""/>
                            Edit
                        </button>
                        <button title="Edit the meal plan" class="header-button" id="delete">
                            <img src="/static/img/delete_24x24.png" height="24" alt=""/>
                            Delete
                        </button>
                    </p>
                </div>
            </div>
        </div>
        <table>
            <thead>
                <tr>
                    <th>Date</th>
                    <th>Serving</th>
                </tr>
            </thead>
            <tbody id="servings"></tbody>
        </table>
        <hr/>
        <p id="notes">{{.Notes}}</p>
        {{template "footer.inc.html"}}
    </div>
</body>
</html>
```

F.3 Other client-side code

F.3.1 Listing of `js/mpajax.js`

```
// The MPAjax object provides functions for interacting with the server over
// Ajax.
var MPAjax = (function() {
    var MPAjax = {};
    // Generic function for performing an Ajax call with the given request data
    // and success callback.
    function doAjax(data, success) {
        $.ajax({
            url: "/api",
            type: "POST",
            dataType: "json",
            data: data,
            error: function(jqXHR, textStatus, errorThrown) {
                console.log("MPAjax HTTP error:");
                console.log("  textStatus = " + textStatus);
                console.log("  errorThrown = " + errorThrown);
                alert("MPAjax error! Check console for more details.");
            },
            success: function(resp, textStatus, jqXHR) {
                if (resp.error) {
                    console.log("MPAjax server error:");
                    console.log("  Message: " + resp.error);
                    alert("MPAjax error! Check console for more details.");
                }
                else {
                    if (MPUtil.nonNull(success)) {
                        success(resp.success);
                    }
                }
            },
        });
    }
    // Fetch the list of meals and pass them to a callback function.
    MPAjax.fetchMealList = function(params, callback) {
        var params = {
            "command": "fetch-meal-list",
            "query": params.query,
```

```
        "sort-column": params.sortColumn,
        "sort-reversed": params.sortReversed,
    };
    doAjax(params, callback);
};
// Toggle the "favourite" status of the meal identified by 'mealID' and pass
// the updated "favourite" status to a callback function.
MPAjax.toggleFavourite = function(mealID, callback) {
    var params = {
        "command": "toggle-favourite",
        "mealid": mealID,
    };
    doAjax(params, callback);
};
// Delete the meal identified by 'mealID' and call the callback function
// when done.
MPAjax.deleteMeal = function(mealID, callback) {
    var params = {
        "command": "delete-meal",
        "mealid": mealID,
    };
    doAjax(params, callback);
};
// Fetch the list of all tags in the database and pass them to a callback
// function.
MPAjax.fetchAllTags = function(callback) {
    var params = {
        "command": "fetch-all-tags",
    };
    doAjax(params, callback);
};
/*
// The first time this function is called, it is identical to 'fetchAllTags'.
// After that, it does nothing.
var tagsFetched = false;
MPAjax.fetchAllTagsOnce = function(callback) {
    console.log(tagsFetched);
    if (!tagsFetched) {
        MPAjax.fetchAllTags(callback);
        tagsFetched = true;
    }
};
*/
// Fetch a list of servings for the meal plan identified by 'mpID' and pass
// them to a callback function.
MPAjax.fetchServings = function(mpID, callback) {
    var params = {
        "command": "fetch-servings",
        "mealplanid": mpID,
    };
    doAjax(params, callback);
};
MPAjax.fetchSuggestions = function(mpID, date, callback) {
    var params = {
        "command": "fetch-suggestions",
        "mealplanid": mpID,
        "date": MPUtil.formatDateJSON(date),
    };
    doAjax(params, callback);
}
MPAjax.updateServing = function(mpID, date, mealID, callback) {
    var params = {
        "command": "update-serving",
        "mealplanid": mpID,
        "date": MPUtil.formatDateJSON(date),
        "mealid": mealID,
    };
    doAjax(params, callback);
};
MPAjax.deleteServing = function(mpID, date, callback) {
    var params = {
        "command": "delete-serving",
        "mealplanid": mpID,
        "date": MPUtil.formatDateJSON(date),
    };
    doAjax(params, callback);
```

```
    };
    MPAjax.updateNotes = function(mpID, notes, callback) {
        var params = {
            "command": "update-notes",
            "mealplanid": mpID,
            "notes": notes,
        };
        doAjax(params, callback);
    };
    MPAjax.fetchMealPlans = function(from, to, callback) {
        var params = {
            "command": "fetch-meal-plans",
            "from": MPUtil.formatDateJSON(from),
            "to": MPUtil.formatDateJSON(to),
        };
        doAjax(params, callback);
    };
    return MPAjax;
})();
```

F.3.2 Listing of `js/mputil.js`

```
// The MPUtil object provides routines that are used multiple times in the
// page-specific JS code. It mostly contains functions for rendering result sets
// as returned by MPAjax into HTML.
var MPUtil = (function() {
    var shortWeekdays = [
        "Sun",
        "Mon",
        "Tue",
        "Wed",
        "Thu",
        "Fri",
        "Sat",
    ];
    var shortMonths = [
        "Jan",
        "Feb",
        "Mar",
        "Apr",
        "May",
        "Jun",
        "Jul",
        "Aug",
        "Sep",
        "Oct",
        "Nov",
        "Dec",
    ];
    var MPUtil = {};
    function zeroPad(str, length) {
        str = "" + str;
        while (str.length < length) {
            str = "0" + str;
        }
        return str;
    }
    function renderNameCell(mt, callback) {
        var nameCell = $("<td>");
        if (MPUtil.nonNull(callback)) {
            $("<a href='#'>").text(mt.meal.name).appendTo(nameCell).click(function(event) {
                event.preventDefault();
                callback(mt);
            });
        }
        else {
            nameCell.text(mt.meal.name);
        }
        return nameCell;
    }
    function renderTagsCell(mt) {
        return $("<td>").text((mt.tags || []).join(", "));
    }
    function renderScoreCell(score) {
        return $("<td>").text(score);
```

```
}
function renderRecipeCell(mt) {
    if (mt.meal.recipe) {
        var button = $("<button title='Open the recipe page listed for this meal' class='action-
            button'><img src='/static/img/open-recipe_16x16.png' height='16' alt=''/></button>")
            .click(function(event) {
                event.preventDefault();
                location.href = mt.meal.recipe;
            });
        return $("<td>").append(button);
    }
    else {
        return $("<td>");
    }
}
function renderFavCell(mt) {
    var toggleFavCallback = function(event) {
        event.preventDefault();
        MPAjax.toggleFavourite(mt.meal.id, function(isFavourite) {
            if (isFavourite) {
                favButton.hide();
                unfavButton.show();
            }
            else {
                unfavButton.hide();
                favButton.show();
            }
        });
    };
    var favButton   = $("<button title='Mark this meal as a favourite' class='action-button'><
        img src='/static/img/favourite_16x16.png' height='16' alt=''/></button>");
    var unfavButton = $("<button title='Remove the favourite mark from this meal' class='action-
        button'><img src='/static/img/unfavourite_16x16.png' height='16' alt=''/></button>");
    favButton.click(toggleFavCallback);
    unfavButton.click(toggleFavCallback);
    if (mt.meal.favourite) {
        favButton.hide();
    }
    else {
        unfavButton.hide();
    }
    return $("<td>").append(favButton).append(unfavButton);
}
function renderEditCell(mt) {
    return $("<td><button title='Edit this meal' class='action-button'><img src='/static/img/
        edit_24x24.png' height='16' alt=''/></button></td>")
        .click(function(event) {
            event.preventDefault();
            location.href = "/meals/" + mt.meal.id + "/edit";
        });
}
function renderDeleteCell(mt, row) {
    return $("<td><button title='Delete this meal from the database' class='action-button'><img
        src='/static/img/delete_24x24.png' height='16' alt=''/></button></td>")
        .click(function(event) {
            event.preventDefault();
            if (confirm("Are you sure you want to delete the meal '" + mt.meal.name + "'?")) {
                MPAjax.deleteMeal(mt.meal.id, function(response) {
                    location.reload();
                });
            }
        });
}
// Renders a single meal/tag result and returns the created <tr> element.
// Used by MPUtil.renderMealList.
function renderMealRow(mt, callback) {
    var row = $("<tr>");
    row.append(renderNameCell(mt, callback).addClass("meal-list-name"));
    row.append(renderTagsCell(mt).addClass("meal-list-tags"));
    row.append(renderRecipeCell(mt).addClass("meal-list-action"));
    row.append(renderFavCell(mt).addClass("meal-list-action"));
    row.append(renderEditCell(mt).addClass("meal-list-action"));
    row.append(renderDeleteCell(mt, row).addClass("meal-list-action"));
    return row;
}
// Renders a single meal/tag result and returns the created <tr> element.
```

```
// Used by MPUtil.renderMealList.
function renderSuggRow(sugg, callback) {
    var row = $("<tr>");
    row.append(renderNameCell(sugg.mt, callback).addClass("sugg-list-name"));
    row.append(renderTagsCell(sugg.mt).addClass("sugg-list-tags"));
    row.append(renderScoreCell(sugg.score).addClass("sugg-list-score"));
    row.append(renderRecipeCell(sugg.mt).addClass("sugg-list-action"));
    row.append(renderFavCell(sugg.mt).addClass("sugg-list-action"));
    row.append(renderEditCell(sugg.mt).addClass("sugg-list-action"));
    row.append(renderDeleteCell(sugg.mt).addClass("sugg-list-action"));
    return row;
}
function renderPage(items, tbody, renderRow, callback, start, count) {
    tbody.empty();
    var end = start + count;
    if (end > items.length) end = items.length;
    var i, row;
    var alt = true;
    for (i = start; i < end; i++) {
        row = renderRow(items[i], callback);
        if (alt) row.addClass("alt");
        row.appendTo(tbody);
        alt = !alt;
    }
}
function updatePageNumCell(pageNumCell, page, numPages) {
    pageNumCell.text("Page " + (page + 1) + " of " + numPages);
}
function renderPaged(items, container, numCols, headerRow, renderRow, callback, highlightPred) {
    items = items || [];
    container.empty();
    if (items.length == 0) {
        container.text("No results to display.");
        return;
    }
    var numPages = Math.floor((items.length + 9) / 10);
    var table = $("<table class='meal-list'>").appendTo(container);
    var thead = $("<thead>").appendTo(table);
    var tbody = $("<tbody>").appendTo(table);
    var navCell = $("<td colspan='"+numCols+"' style='padding-bottom: 20px'>").appendTo($("<tr>"
        ).appendTo(thead));
    headerRow.appendTo(thead);
    var page = 0;
    var highlightIndex = null;
    if (MPUtil.nonNull(highlightPred)) {
        var i;
        for (i = 0; i < items.length; i++) {
            if (highlightPred(items[i])) {
                page = Math.floor(i / 10);
                highlightIndex = i % 10;
                break;
            }
        }
    }
    var prevCell    = $("<div class='table-nav table-nav-left'>").appendTo(navCell);
    var pageNumCell = $("<div class='table-nav table-nav-center'>").appendTo(navCell);
    var nextCell    = $("<div class='table-nav table-nav-right'>").appendTo(navCell);
    $("<button title='Navigate to the first page of results'><img src='/static/img/first_24x24.
        png' height='16' alt='First'/></button>")
        .appendTo(prevCell)
        .click(function(event) {
            event.preventDefault();
            page = 0;
            updatePageNumCell(pageNumCell, page, numPages);
            renderPage(items, tbody, renderRow, callback, page * 10, 10);
        });
    $("<button title='Navigate to the previous page of results'><img src='/static/img/prev_24x24
        .png' height='16' alt='Prev'/></button>")
        .appendTo(prevCell)
        .click(function(event) {
            event.preventDefault();
            if ((page - 1) >= 0) {
                page -= 1;
            }
            updatePageNumCell(pageNumCell, page, numPages);
            renderPage(items, tbody, renderRow, callback, page * 10, 10);
```

```
        });
    $("<button title='Navigate to the next page of results'><img src='/static/img/next_24x24.png
        ' height='16' alt='Next'/></button>")
        .appendTo(nextCell)
        .click(function(event) {
            event.preventDefault();
            if ((page + 1) < numPages) {
                page += 1;
            }
            updatePageNumCell(pageNumCell, page, numPages);
            renderPage(items, tbody, renderRow, callback, page * 10, 10);
        });
    $("<button title='Navigate to the last page of results'><img src='/static/img/last_24x24.png
        ' height='16' alt='Last'/></button>")
        .appendTo(nextCell)
        .click(function(event) {
            event.preventDefault();
            page = numPages - 1;
            updatePageNumCell(pageNumCell, page, numPages);
            renderPage(items, tbody, renderRow, callback, page * 10, 10);
        });
    updatePageNumCell(pageNumCell, page, numPages);
    renderPage(items, tbody, renderRow, callback, page * 10, 10);
    if (MPUtil.nonNull(highlightIndex)) {
        var row = $(tbody.find("tr")[highlightIndex]);
        var bg = row.css("background");
        row.css("background", "#0f0");
        row.animate({
            backgroundColor: bg,
        }, 1000);
    }
};
// Takes a list of meal/tags results (as returned by MPAjax.fetchMealList)
// and renders them to a table created inside 'container'. 'callback', if
// not null, is a function that will be called when the meal name is clicked.
// It is passed the meal/tags object.
MPUtil.renderMealList = function(mts, container, callback) {
    var headerRow = $("<tr><th class='meal-list-name'>Name</th><th class='meal-list-tags'>Tags</
        th><th colspan='4' class='meal-list-actions'>Actions</th></tr>");
    renderPaged(mts, container, 6, headerRow, renderMealRow, callback, null);
};
MPUtil.renderMealListHighlight = function(mts, container, highlightID, callback) {
    var headerRow = $("<tr><th class='meal-list-name'>Name</th><th class='meal-list-tags'>Tags</
        th><th colspan='4' class='meal-list-actions'>Actions</th></tr>");
    var highlightPred = function(mt) {return mt.meal.id == highlightID};
    renderPaged(mts, container, 6, headerRow, renderMealRow, callback, highlightPred);
};
MPUtil.renderSuggestions = function(suggs, container, callback) {
    var headerRow = $("<tr><th class='sugg-list-name'>Name</th><th class='sugg-list-tags'>Tags</
        th><th class='sugg-list-score'>Score</th><th colspan='4' class='sugg-list-actions'>
        Actions</th></tr>")
    renderPaged(suggs, container, 7, headerRow, renderSuggRow, callback, null);
};
// Takes a list of tags (as returned by MPAjax.fetchAllTags) and renders
// them to the <select> tag 'container'.
MPUtil.renderExistingTagsList = function(tags, container) {
    tags = tags || [];
    var i, tag;
    for (i = 0; i < tags.length; i++) {
        tag = tags[i];
        $("<option>").val(tag).text(tag).appendTo(container);
    }
};
MPUtil.formatMonthHumanReadable = function(date) {
    return shortMonths[date.getMonth()] + " " + date.getFullYear();
};
MPUtil.formatDateHumanReadable = function(date) {
    return shortWeekdays[date.getDay()] + " " + date.getDate() + " " + shortMonths[date.getMonth
        ()];
};
MPUtil.formatDateJSON = function(date) {
    return zeroPad(date.getFullYear(), 4) + "-" + zeroPad(date.getMonth() + 1, 2) + "-" +
        zeroPad(date.getDate(), 2);
};
MPUtil.parseDatepickerDate = function(str) {
    parts = str.split("/");
```

```
        if (parts.length < 3 || 1*parts[2] == NaN || 1*parts[1] == NaN || 1*parts[0] == NaN) {
            return null;
        }
        return new Date(parts[2], parts[1]-1, parts[0]);
    };
    MPUtil.nonNull = function(value) {
        return typeof value !== "undefined" && value !== null;
    };
    MPUtil.round1dp = function(x) {
        return Math.round(x * 10) / 10;
    };
    return MPUtil;
})();
```

F.3.3 Listing of js/meallistview.js

```
var MealResult = (function() {
    var MealResult = function(mt, score) {
        this.id = mt.meal.id;
        this.name = mt.meal.name;
        this.recipe = mt.meal.recipe;
        this.favourite = mt.meal.favourite;
        this.tags = mt.tags;
        this.score = score;
    };
    MealResult.fetchMealList = function(params, callback) {
        MPAjax.fetchMealList(params, function(mts) {
            mts = mts || [];
            var i, results = [];
            for (i = 0; i < mts.length; i++) {
                results.push(new MealResult(mts[i], null));
            }
            callback(results);
        });
    };
    MealResult.fetchSuggestions = function(mpID, date, callback) {
        MPAjax.fetchSuggestions(mpID, date, function(suggs) {
            suggs = suggs || [];
            var i, results = [];
            for (i = 0; i < suggs.length; i++) {
                results.push(new MealResult(suggs[i].mt, suggs[i].score));
            }
            callback(results);
        });
    };
    MealResult.prototype.hasScore = function() {
        return MPUtil.nonNull(this.score);
    };
    return MealResult;
})();
var MealListViewColumns = (function() {
    var o = {};
    o.NameColumn = function(className) {
        this.view = null;
        this.className = className;
    };
    o.NameColumn.prototype.renderHeader = function(row) {
        /*
        var view = this.view;
        var th = $("<th></th>").addClass(this.className).appendTo(row);
        $("<a href='#'>Name</a>").appendTo(th).click(function() {
            view.sort("name");
        });
        */
        $("<th>Name</th>").addClass(this.className).appendTo(row);
    };
    o.NameColumn.prototype.renderData = function(row, item) {
        var cell = $("<td></td>").addClass(this.className).appendTo(row);
        if (MPUtil.nonNull(this.view.itemCallback)) {
            var view = this.view;
            var link = $("<a href='#'></a>").text(item.name).appendTo(cell).click(function(event) {
                event.preventDefault();
                view.itemCallback.call(view, item);
            });
        }
```

```
    else {
        cell.text(item.name);
    }
};
o.TagsColumn = function(className) {
    this.view = null;
    this.className = className;
};
o.TagsColumn.prototype.renderHeader = function(row) {
    $("<th>Tags</th>").addClass(this.className).appendTo(row);
};
o.TagsColumn.prototype.renderData = function(row, item) {
    var tagsString = (item.tags || []).join(", ");
    $("<td></td>").text(tagsString).addClass(this.className).appendTo(row);
};
o.ScoreColumn = function(className) {
    this.view = null;
    this.className = className;
};
o.ScoreColumn.prototype.renderHeader = function(row) {
    /*
    var view = this.view;
    var th = $("<th></th>").addClass(this.className).appendTo(row);
    $("<a href='#'>Score</a>").appendTo(th).click(function() {
        view.sort("score");
    });
    */
    return $("<th>Score</th>").addClass(this.className).appendTo(row);
};
o.ScoreColumn.prototype.renderData = function(row, item) {
    var scoreStr = "" + MPUtil.round1dp(item.score * 9 + 1);
    if (scoreStr.indexOf(".") < 0) {
        scoreStr += ".0";
    }
    $("<td></td>").text(scoreStr).addClass(this.className).appendTo(row);
};
o.ActionsColumn = function(className) {
    this.view = null;
    this.className = className;
};
o.ActionsColumn.prototype.renderHeader = function(row) {
    $("<th>Actions</th>").addClass(this.className).appendTo(row);
};
o.ActionsColumn.prototype.renderData = function(row, item) {
    var cell = $("<td></td>").addClass(this.className).appendTo(row);
    this.renderRecipeButton(cell, item);
    this.renderFavButton(cell, item);
    this.renderEditButton(cell, item);
    this.renderDeleteButton(cell, item);
};
o.ActionsColumn.prototype.renderRecipeButton = function(cell, item) {
    var cell = $("<div class='action-button-container'></div>").addClass(this.
        individualClassName).appendTo(cell);
    var button = $("<button title='Open the recipe page listed for this meal' class='action-
        button'><img src='/static/img/open-recipe_16x16.png' height='16' alt=''/></button>");
    if (item.recipe) {
        button.appendTo(cell).click(function(event) {
            event.preventDefault();
            location.href = item.recipe;
        });
    }
    else {
        button.attr("disabled", "disabled");
    }
};
o.ActionsColumn.prototype.renderFavButton = function(cell, item) {
    var cell = $("<div class='action-button-container'></div>").addClass(this.
        individualClassName).appendTo(cell);
    var favButton   = $("<button title='Mark this meal as a favourite' class='action-button'><
        img src='/static/img/favourite_16x16.png' height='16' alt=''/></button>");
    var unfavButton = $("<button title='Remove the favourite mark from this meal' class='action-
        button'><img src='/static/img/unfavourite_16x16.png' height='16' alt=''/></button>");
    var toggleFavCallback = function(event) {
        event.preventDefault();
        MPAjax.toggleFavourite(item.id, function(isFavourite) {
            item.favourite = isFavourite;
```

```
                if (isFavourite) {
                    favButton.hide();
                    unfavButton.show();
                }
                else {
                    unfavButton.hide();
                    favButton.show();
                }
            });
        };
        favButton.appendTo(cell).click(toggleFavCallback);
        unfavButton.appendTo(cell).click(toggleFavCallback);
        if (item.favourite) {
            favButton.hide();
        }
        else {
            unfavButton.hide();
        }
    };
    o.ActionsColumn.prototype.renderEditButton = function(cell, item) {
        var cell = $("<div class='action-button-container'></div>").addClass(this.
            individualClassName).appendTo(cell);
        var button = $("<button title='Edit this meal' class='action-button'><img src='/static/img/
            edit_24x24.png' height='16' alt=''/></button>");
        button.appendTo(cell).click(function(event) {
            event.preventDefault();
            location.href = "/meals/" + item.id + "/edit";
        });
    };
    o.ActionsColumn.prototype.renderDeleteButton = function(cell, item) {
        var cell = $("<div class='action-button-container'></div>").addClass(this.
            individualClassName).appendTo(cell);
        var button = $("<button title='Delete this meal from the database' class='action-button'><
            img src='/static/img/delete_24x24.png' height='16' alt=''/></button>");
        var view = this.view;
        button.appendTo(cell).click(function(event) {
            event.preventDefault();
            if (confirm("Are you sure that you want to delete the meal '" + item.name + "'?")) {
                MPAjax.deleteMeal(item.id, function(response) {
                    view.deleteItemByID(item.id);
                });
            }
        });
    };
    return o;
})();
var MealListView = (function() {
    var MealListView = function(parent) {
        this.parent = parent;
        this.items = [];
        this.numPages = 0;
        this.currentPage = 0;
        this.columns = [];
        this.itemCallback = null;
        this.fetchDataCallback = null;
        this.deleteCallback = null;
        this.tbody = null;
        this.pageNumSpan = null;
        this.numPagesSpan = null;
        this.dataParams = {
            query: "",
            sortColumn: "",
            sortReversed: false,
        };
    };
    MealListView.prototype.setData = function(items) {
        this.items = items;
        this.currentPage = 0;
        this.touchData();
    };
    MealListView.prototype.touchData = function() {
        this.numPages = Math.floor((this.items.length + 9) / 10);
        this.renderCurrentPage();
    };
    MealListView.prototype.getCurrentPage = function() {
        return this.currentPage;
```

```
};
MealListView.prototype.setCurrentPage = function(p) {
    this.currentPage = p;
    this.renderCurrentPage();
};
MealListView.prototype.incrCurrentPage = function(amt) {
    this.currentPage += amt;
    this.renderCurrentPage();
};
MealListView.prototype.lookup = function(id) {
    var i;
    for (i = 0; i < this.items.length; i++) {
        if (this.items[i].id == id) {
            return i;
        }
    }
    return null;
};
MealListView.prototype.deleteItemByID = function(id) {
    var idx = this.lookup(id);
    if (MPUtil.nonNull(idx)) {
        this.deleteItemByIndex(idx);
    }
};
MealListView.prototype.deleteItemByIndex = function(idx) {
    if (MPUtil.nonNull(this.deleteCallback)) {
        this.deleteCallback.call(this, this.items[idx]);
    }
    this.items.splice(idx, 1);
    this.touchData();
};
MealListView.prototype.highlightItemByID = function(id) {
    var idx = this.lookup(id);
    if (MPUtil.nonNull(idx)) {
        this.highlightItemByIndex(idx);
    }
};
MealListView.prototype.highlightItemByIndex = function(idx) {
    var newCurrentPage = Math.floor(idx / 10);
    if (this.currentPage != newCurrentPage) {
        this.setCurrentPage(newCurrentPage);
    }
    var row = $(this.tbody.find("tr")[idx % 10]);
    var bg = row.css("background");
    row.css("background", "orange");
    row.animate({backgroundColor: bg}, 1000);
};
MealListView.prototype.setItemCallback = function(cb) {
    this.itemCallback = cb;
};
MealListView.prototype.setFetchDataCallback = function(cb) {
    this.fetchDataCallback = cb;
};
MealListView.prototype.setDeleteCallback = function(cb) {
    this.deleteCallback = cb;
};
MealListView.prototype.addColumn = function(col) {
    col.view = this;
    this.columns.push(col);
};
MealListView.prototype.sort = function(column) {
    if (this.dataParams.sortColumn == column) {
        this.dataParams.sortReversed = !this.dataParams.sortReversed;
    }
    else {
        this.dataParams.sortColumn = column;
        this.dataParama.sortReversed = false;
    }
};
MealListView.prototype.render = function() {
    this.parent.empty();
    /*
    if (this.items.length == 0) {
        this.parent.text("No results to display.");
        return;
    }
```

```
    */
    this.renderSearch(this.parent);
    this.renderNav(this.parent);
    var table = $("<table style='width: 100%; table-layout: fixed'></table>").appendTo(this.
        parent);
    var thead = $("<thead></thead>").appendTo(table);
    var tbody = $("<tbody></tbody>").appendTo(table);
    var headerRow = $("<tr></tr>").appendTo(thead);
    var i;
    for (i = 0; i < this.columns.length; i++) {
        this.columns[i].renderHeader(headerRow);
    }
    this.tbody = tbody;
    if (MPUtil.nonNull(this.fetchDataCallback)) {
        this.fetchData();
    }
};
MealListView.prototype.fetchData = function() {
    var view = this;
    this.fetchDataCallback.call(this, this.dataParams, function(items) {
        view.setData(items);
    });
};
MealListView.prototype.renderCurrentPage = function() {
    if (this.currentPage >= this.numPages) {
        this.currentPage = this.numPages - 1;
    }
    if (this.currentPage < 0) {
        this.currentPage = 0;
    }
    this.tbody.empty();
    this.pageNumSpan.text(this.currentPage + 1);
    this.numPagesSpan.text(this.numPages);
    var start = this.currentPage * 10;
    var end = start + 10;
    if (end > this.items.length) {
        end = this.items.length;
    }
    var i;
    for (i = start; i < end; i++) {
        this.renderItem(this.items[i]);
    }
};
MealListView.prototype.renderItem = function(item) {
    var row = $("<tr></tr>").appendTo(this.tbody);
    var i;
    for (i = 0; i < this.columns.length; i++) {
        this.columns[i].renderData(row, item);
    }
};
MealListView.prototype.renderSearch = function(parent) {
    var container = $("<div class='table-search-container'></div>").appendTo(parent);
    var img = $("<img src='/static/img/loading.gif' height='16' alt='Searching...' style='margin
        -right: 10px'/>").appendTo(container).hide();
    var input = $("<input type='text' placeholder='Type to search...' width='30'/>").appendTo(
        container);
    var tid = null;
    var view = this;
    input.keydown(function() {
        if (MPUtil.nonNull(tid)) {
            window.clearTimeout(tid);
        }
        tid = window.setTimeout(function() {
            tid = null;
            img.show();
            view.dataParams.query = input.val();
            if (MPUtil.nonNull(view.fetchDataCallback)) {
                view.fetchDataCallback.call(view, view.dataParams, function(items) {
                    view.setData(items);
                    img.hide();
                });
            }
        }, 1200);
    });
};
MealListView.prototype.renderNav = function(parent) {
```

```
        var nav = $("<div class='table-nav-container'></div>").appendTo(parent);
        var left = $("<div class='table-nav table-nav-left'></div>").appendTo(nav);
        var center = $("<div class='table-nav table-nav-center'></div>").appendTo(nav);
        var right = $("<div class='table-nav table-nav-right'></div>").appendTo(nav);
        var view = this;
        var firstButton = $("<button title='Navigate to the first page of results'><img src='/static
            /img/first_24x24.png' height='16' alt='First'/></button>");
        firstButton.appendTo(left).click(function(event) {
            event.preventDefault();
            view.setCurrentPage(0);
        });
        var prevButton = $("<button title='Navigate to the previous page of results'><img src='/
            static/img/prev_24x24.png' height='16' alt='Prev'/></button>");
        prevButton.appendTo(left).click(function(event) {
            event.preventDefault();
            view.incrCurrentPage(-1);
        });
        var nextButton = $("<button title='Navigate to the next page of results'><img src='/static/
            img/next_24x24.png' height='16' alt='Next'/></button>");
        nextButton.appendTo(right).click(function(event) {
            event.preventDefault();
            view.incrCurrentPage(1);
        });
        var lastButton = $("<button title='Navigate to the last page of results'><img src='/static/
            img/last_24x24.png' height='16' alt='Last'/></button>");
        lastButton.appendTo(right).click(function(event) {
            event.preventDefault();
            view.setCurrentPage(view.numPages - 1);
        });
        center.html("Page <span id='page-num'></span> of <span id='num-pages'></span>");
        this.pageNumSpan = center.find("#page-num");
        this.numPagesSpan = center.find("#num-pages");
    };
    return MealListView;
})();
```

F.3.4 Listing of css/screen.css

```
/* User-defined CSS */
button.header-button {
    font-size: large;
    margin-top: 20px;
}
button.homepage-button {
    font-size: x-large;
    padding: 10px;
    width: 300px;
    margin-bottom: 5px;
}
button.action-button {
    margin-bottom: 3px;
}
.error {
    font-weight: bold;
    color: red;
}
.homepage-cell {
    text-align: center;
}
.row {
    margin-bottom: 20px;
}
td, th {
    padding-right: 20px;
    padding-top: 5px;
    padding-bottom: 5px;
}
th {
    text-align: center;
}
table.tags-list {
    border: 1px solid #eee;
}
table.tags-list tbody tr td {
    padding-top: 5px;
```

```
    padding-left: 10px;
    padding-bottom: 5px;
}
button.remove-tag {
    font-size: small;
}
ul.navigation {
    border-bottom: 1px solid #eee;
}
ul.navigation li {
    display: inline;
    background-image: url("/static/img/nav-sep.png");
    background-color: #ffffff;
    background-repeat: no-repeat;
    background-position: 3px 2px;
    background-size: 12px 12px;
    padding-left: 20px;
}
ul.navigation li.home {
    padding-left: 0px;
    background-image: none;
}
table.mplist {
    border-collapse: separate;
    border-spacing: 0px 10px;
}
table.mplist thead tr th, table.mplist tbody tr td {
    padding: 0px;
    font-size: large;
    text-align: center;
}
.mpstart div, .mpmid div, .mpend div {
    border-top: 1px solid #f00;
    border-bottom: 1px solid #f00;
    padding-top: 1px;
    padding-bottom: 1px;
    background: #fee;
}
.mpstart div {
    margin-left: 6px;
    border-left: 1px solid #f00;
    padding-left: 5px;
    padding-right: 13px;
}
.mpmid div {
    padding-left: 13px;
    padding-right: 13px;
}
.mpnone div {
    padding-left: 13px;
    padding-right: 13px;
}
.mpend div {
    padding-left: 13px;
    padding-right: 5px;
    border-right: 1px solid #f00;
    margin-right: 6px;
}
.mpstart.hover div, .mpmid.hover div, .mpend.hover div {
    border-top: 2px solid #f00;
    border-bottom: 2px solid #f00;
    padding-top: 0px;
    padding-bottom: 0px;
    background: #fdd;
}
.mpstart.hover div {
    border-left: 2px solid #f00;
    padding-left: 4px;
}
.mpend.hover div {
    border-right: 2px solid #f00;
    padding-right: 4px;
}
.outside-key-month {
    color: #999;
}
```

```
.today {
    font-weight: bold;
    color: #009;
}
.footer {
    font-size: x-small;
    color: #666;
}
.meal-list {
    table-layout: fixed;
}
.meal-list-name {
    width: 45%;
}
.meal-list-tags {
    width: 40%;
}
.meal-list-actions {
    width: 15%;
}
td.meal-list-actions {
    text-align: right;
}
.sugg-list {
    table-layout: fixed;
}
.sugg-list-name {
    width: 35%;
}
.sugg-list-tags {
    width: 35%;
}
.sugg-list-score {
    width: 15%;
}
.sugg-list-actions {
    width: 15%;
}
td.sugg-list-actions {
    text-align: right;
}
div.action-button-container {
    display: inline;
}
.table-search-container {
    text-align: right;
    margin-bottom: 10px;
}
.table-nav-container {
    width: 100%;
}
.table-nav {
    float: left;
    margin-top: 10px;
    margin-bottom: 20px;
    height: 40px;
    border-bottom: 1px solid #eee;
}
.table-nav-left {
    width: 20%;
    text-align: left;
}
.table-nav-center {
    width: 60%;
    text-align: center;
}
.table-nav-right {
    width: 20%;
    text-align: right;
}
tr.alt {
    background: #eef;
}
#page-num, #num-pages {
    font-weight: bold;
}
```